National Information Exchange Model Model Package Description Specification

Version 3.0alpha7

February 5, 2014

NIEM Technical Architecture Committee (NTAC)

Contents

Abstract

This document specifies normative rules and non-normative guidance for building Model Package Descriptions (MPDs) that conform to the National Information Exchange Model (NIEM) version 3.0.

Status

This document is a draft of the specification for NIEM Model Package Descriptions (MPDs). It represents the design that has evolved from the collaborative work of the NIEM Business Architecture Committee (NBAC) and the NIEM Technical Architecture Committee (NTAC) and their predecessors.

This specification is a product of the NIEM Program Management Office (PMO).

Email comments on this specification to `niem-comments@lists.gatech.edu`.

Remaining work

1. Schematron rules for MPD Catalog (associated resource tool kit?)
2. Place conformance target notation on all rules.
3. Design to consolidate HasElement, HasDocumentElement, etc. in MPD catalog XSD.
4. Identify convenient classification of rules based on conformance targets.
5. Definition(s) for "resolve".
6. Better definitions for schema sets.
7. Mandatory ToC entries in mpd-catalog.xml (release, CU, DU, IEPD, EIEM dependent)
8. Appendix for Example IEP (from Cursor on Target)
9. QA and validity checks on MPD catalog schema (Appendix A)
10. Recheck acronymns/abbreviations

NOTE: Search on "TBD" to find incomplete items.

1. Introduction

This specification assumes familiarity with the National Information Exchange Model (NIEM), its basic concepts, architecture, processes, design rules, and general conformance rules. For novices to NIEM, the recommended reading list includes:

- Introduction to the National Information Exchange Model **[NIEM Introduction]**
- NIEM Concept of Operations **[NIEM Concept of Operations]**
- NIEM Naming and Design Rules **[NIEM NDR]**
- NIEM High-Level Version Architecture **[NIEM High-Level Version Architecture]**
- NIEM High-Level Tool Architecture **[NIEM High-Level Tool Architecture]**
- NIEM Conformance **[NIEM Conformance]**
- NIEM User Guide **[NIEM User Guide]**
- NIEM Business Information Exchange Components **[NIEM BIEC]**
- NIEM Implementation Guidelines **[NIEM Implementation Guidance]**

The foregoing NIEM documents are available at `http://reference.niem.gov/niem/`. See **[NIEM Implementation Guidance]** for NIEM Implementation Guidelines.

Those knowledgeable of NIEM should be familiar with the **[NIEM NDR]**, **[NIEM High-Level**

Version Architecture], **[NIEM Conformance]**, and **[NIEM BIEC]**.

This MPS Specification v3.0 uses and is a peer to the NIEM Naming and Design Rules (NDR) **[NIEM NDR]**. It supersedes IEPD guidance previously published in Requirements for a NIEM IEPD **[NIEM IEPD Requirements]** and the NIEM User Guide **[NIEM User Guide]**. The NIEM User Guide remains a good source for understanding the process of building Information Exchange Package Documentation (IEPD). It also supersedes both MPD Specification v1.0 and v1.1.

1.1. Background

Many fundamental concepts, processes, and products in the NIEM generally involve aggregating electronic files into logical sets that serve a specific purpose. Examples of such sets include, but in the future may not necessarily be limited to, a NIEM release, core update (CU), domain update (DU), Information Exchange Package Documentation (IEPD), and Enterprise Information Exchange Model (EIEM). Each of these examples is a NIEM Model Package Description (MPD).

[Definition: Model Package Description (MPD)]

A set of related W3C XML Schema documents and other supporting files organized as one of the five classes of NIEM schema sets:

- Release (major, minor, or micro).
- Domain update (DU) to a release.
- Core update (CU) to a release.
- Information Exchange Package Documentation (IEPD).
- Enterprise Information Exchange Model (EIEM).

An MPD is self-documenting and provides sufficient normative and non-normative information to allow technical personnel to understand how to use or implement it. An MPD is packaged as a **[PKZIP]** archive file.

A key NIEM concept used throughout this specification is *data component*.

[Definition: data component]

An XML Schema type or attribute group definition; or an XML Schema element or attribute declaration.

An MPD is a normative specification for XML data components in the format of the World Wide Web Consortium (W3C) XML Schema Definition Language **[W3C XML Schema Datatypes]**, **[W3C XML Schema Structures]**. MPD schema documents either (1) define the semantics and structure for NIEM reusable data components, or (2) define implementable NIEM exchange instance documents in W3C Extensible Markup Language (XML) **[W3-XML]**.

An MPD is ready to publish and use when it has been properly packaged with the schemas, documentation, and supplemental files needed to understand how to use and implement it. MPD

content design, development, and assembly may be difficult and time-consuming, especially if done manually. Software tools have been shown to significantly reduce the complexity of designing, constructing, changing, and managing MPDs. In order to reduce ambiguity and to facilitate interoperable and effective tool support, this baseline specification imposes some degree of consistency on the terminology, syntax, semantics, and composition of MPDs.

1.2. Purpose

This document is a normative specification for the various kinds of NIEM MPDs. The rules and guidance herein are designed to encourage and facilitate NIEM use and tools by balancing consistency, simplicity, and flexibility. Consistency and simplicity make MPDs easy to design correctly, build rapidly, and find easily (for reuse or adaptation). Consistency also facilitates tool support. Flexibility enables more latitude to design and tailor MPDs for complex data exchange requirements. As such, this document does not necessarily prescribe mandates or rules for all possible situations or organizational needs. If an organization determines it should impose additional constraints or requirements on its IEPDs beyond those specified in this document (for example, mandating a normative set of business requirements or a domain model within IEPD documentation), then it is free to do so, as long as no conflicts exist with this MPD Specification or the **[NIEM NDR]**.

This document defines terminology; identifies required and optional (but common) artifacts; defines metadata; specifies normative rules, schemes, and syntax; provides non-normative guidance; and as needed, refers to other related NIEM specifications for more detail.

1.3. Scope

This specification applies to information exchange definitions and release products that employ the data component definitions and declarations in NIEM Core and Domains. It also applies to the NIEM release products and their associated updates. In particular, this version of this document applies to the following MPDs:

- NIEM releases (including major, minor, and micro releases).
- NIEM domain updates (DU) **[NIEM Domain Update Specification]**. (Note these are NOT the same as the NIEM domain schemas that are part of numbered releases).
- Core updates (CU) to NIEM releases.
- Information Exchange Package Documentation (IEPD) that define NIEM data exchanges.
- Enterprise Information Exchange Model (EIEM) from which one or more NIEM IEPDs can be built or based.

In the future, as required, other types of MPDs may be added to this list.

At any point in time, an incomplete MPD will be in some state of development. This specification is applicable to such developing products in that it establishes validity standards for MPDs in progress, as well as completeness standards for MPDs that reach a final, published, production-quality state. In turn, tool vendors should be able to build, adapt, and/or integrate software tools that will assist in MPD development and assembly from raw parts to finished product.

NIEM is a data layer for an information architecture. Files in an MPD generally define XML Schema types and declare XML elements and attributes to use in payloads for information exchanges. While an MPD may also contain files from layers beyond the data layer, this speciication is not intended to

define details of other architectural layers. Such files are generally present only to provide additional context, understanding, or assistance for implementing the exchange of payloads.

Authoritative sources are not required to revise MPDs that exist before this specification becomes effective. However, they are always encouraged to consider revising MPDs to meet this specification, especially when making other significant changes.

1.4. Audience

The following groups should review and adhere to this specification:

- The NIEM release manager who is responsible to integrate and publish NIEM releases and core updates.
- NIEM domain stewards and technical representatives who develop and publish domain updates.
- NIEM IEPD developers and implementers.
- NIEM-aware tool developers and vendors.
- Organizations that intend to develop an EIEM.
- Individuals or groups responsible to review and approve MPDs.

2. Concepts and Terminology

The presentation of concepts and terms in this section is sequenced for understanding. Each subsection builds upon previous ones. This section concludes with an explanation of each of the five MPD classes and a summary of their similarities and differences.

2.1. Key Words for Requirement Levels

Within normative content rules and definitions, the key words MUST, MUST NOT, SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, RECOMMENDED, REQUIRED, and OPTIONAL in this document are to be interpreted as described in **[RFC2119 Key Words]**.

2.2. Character Case Sensitivity

This specification imposes many constraints on the syntax for identifiers, names, labels, strings, etc. In all cases, unless otherwise explicitly noted, syntax is case sensitive. In particular, XML files in appendices that define particular artifacts, transformations, and examples are case sensitive.

Also, note that as a general principle, lower case characters are used whenever such will not conflict with the **[NIEM NDR]**.

2.3. Artifacts

MPDs are generally composed of files and file sets grouped for a particular purpose. Each file is referred to as an *artifact*, and each logical set of such files is called an *artifact set*.

[Definition: artifact]

A single file with a defined purpose.

[Definition: artifact set]

A collection of [artifacts] logically grouped for a defined purpose.

An MPD is itself a set of artifacts, the purpose for which is to define and document the intended use of the MPD. While the kernel of an MPD is its XML schema document (XSD) artifacts, there are also other kinds of MPD artifacts. These may include HTML (or XML converted to HTML for display), text, or graphic files used for human-readable documentation. An MPD may also have artifacts intended to help assist in or accelerate the use and implementation of the MPD. For example, these may be XML, UML, or binary files that are inputs to or outputs from software tools used to build, generate, or edit the MPD or its schema document artifacts. Appendix C, *MPD Artifacts*, below, contains a listing of mandatory and common optional artifacts for the five types of MPDs. Common types of artifacts are described in more detail in subsequent sections.

2.4. Schema Document and Namespace Correspondence in NIEM

To simplify automatic schema processing and reduce the potential for confusion and error, **[NIEM NDR]** principles state that each NIEM-conformant namespace SHOULD be defined by exactly one reference schema document. To support this concept, the **[NIEM NDR]** disallows the use of `xs:include`, and mandates the use of the `xs:schema/@targetNamespace` attribute in NIEM-conformant schema documents.

So, (1) each NIEM namespace is defined by a single NIEM-conformant schema document, and (2) each NIEM-conformant schema document declares a target namespace. NIEM does not permit schema documents without target namespaces, unless they are from sources outside of NIEM.

2.5. Harmonization

Harmonization is a process that NIEM governance committees and domain stewards iteratively apply to NIEM content (specifically, its semantics, structure, and relationships) during the preparation of a NIEM major or minor release. On a more restricted scale a domain steward harmonizes his/her own content (schema documents) in preparation for a [domain update] MPD. Multiple domain stewards may collaborate in a coordinated [domain update]. In this case, to the extent possible, harmonization may be applied across the content of all the collaborating domains. Harmonization results in model change and evolution with the intent of removing semantic duplication and overlap while improving representational quality and usability.

[Definition: harmonization]

Given a data model, harmonization is the process of reviewing its existing data definitions and declarations; reviewing how it structures and represents data; integrating new data components; and refactoring data components as necessary to remove (or reduce to the maximum extent) semantic duplication and/or semantic overlap among all data structures

and definitions resulting in representational quality improvements.

2.6. XML Validation

A discussion of XML validation requires a basic understanding of basic XML terminology. The following definitions are necessary.

[Definition: XML document]

A document in XML format as defined by **[W3-XML]**, §2.

[Definition: schema component]

A *schema component* is as defined by **[W3C XML Schema Structures]** §2.2, "XML Schema Abstract Data Model", which states:

> **Schema component** is the generic term for the building blocks that comprise the abstract data model of the schema.

[Definition: XML Schema]

An *XML Schema* is as defined by **[W3C XML Schema Structures]** §2.2, "XML Schema Abstract Data Model", which states:

> An **XML Schema** is a set of schema components.

[Definition: XML schema validation]

The process of checking an [XML document] to confirm that it is both well-formed and *valid*, in that it follows the structure defined by an associated [XML Schema]. A well-formed document follows the basic syntactic rules of XML, which are the same for all XML documents.

[Definition: XML schema document]

A physical (file) representation of part or all of an [XML Schema]. One or more *XML schema documents* are used to assemble [schema components] into an [XML Schema].

This specification often refers to the process of [XML schema validation], that is, validation of an instance XML document to confirm it adheres to the structure defined by a particular [XML Schema]. Generally, this should occur periodically during and after design time to ensure the conformance and quality of an information exchange definition (i.e., [XML schema documents]) and associated instance XML documents. However, local architecture or policy may dictate the need to validate more often, and in some cases may require runtime validation.

XML schema document sets that define a NIEM information exchange must be authoritative. Application developers may use other schemas (e.g., constraint or Schematron schema documents) for various purposes, but for the purposes of determining NIEM conformance, the authoritative reference schema documents (NIEM releases) are relevant. This does not mean that XML validation must be performed on all instance XML documents as they are served or consumed; only that the instance XML documents validate if and when XML validation is performed. Therefore, even when validation is not performed, instance XML documents must be valid against the XML schema that is assembled from XML schema document sets that specify these instance XML documents.

2.7. Reference Schema Documents

A NIEM *reference schema document* is a schema document that is intended to be the authoritative definition of business semantics for components within its target namespace. Reference schema documents include the NIEM Core schema documents, NIEM domain schema documents, and NIEM domain update schema documents. The normative definition for a reference schema document and applicable conformance rules are found in the **[NIEM NDR]**. The definition is repeated here:

[Definition: reference schema document]

An XML Schema document that meets all of the following criteria:

- It is a conformant schema document.
- It is explicitly designated as a reference schema document. This may be declared by an MPD catalog or by a tool-specific mechanism outside the schema document.
- It provides the broadest, most fundamental definitions of components in its namespace.
- It provides the authoritative definition of business semantics for components in its namespace.
- It is intended to serve as the basis for components in IEPD and EIEM schema documents, including subset, constraint, and extension schema documents.

See also [reference schema document set].

[Definition: reference schema document set]

A set of related reference schema documents, such as a NIEM release. See also [reference schema document].

The **[NIEM NDR]** conformance rules for reference schema documents are generally stricter than those for other classes of NIEM-conformant schema documents. For example, they are required to employ an `xs:annotation` with child elements `xs:documentation` and `xs:appinfo` that encapsulate semantic information for each XML element and attribute declaration, and type definition.

NIEM reference schemas are very uniform in their structure. As they are the primary definitions for data components, they do not need to restrict other data definitions, and they are not allowed to use XML Schema's complex type restriction mechanisms.

2.8. Coherence of Schema Document Sets

A NIEM release is always a *coherent* set of reference schema documents in which multiple versions of semantically identical types or properties do not exist; and all types and properties are uniquely defined and declared. Each numbered release has been harmonized, tested, and carefully reviewed by NIEM governance committees in order to eliminate semantic duplication. The **[NIEM High-Level Version Architecture]** defines a *coherent schema document set* as one that has the following properties:

> **[Definition: schema document set coherence]**
>
> A schema document set is coherent when it has the following properties: (1) the set does not refer to a schema document outside the set (i.e., the set is closed), and (2) the set does not include two different versions of the same component in an incompatible way.

Consider the following simple example of incoherence in the figures below. Consider Figure 2-1, *Incoherent schema set - not closed*, below, in which Justice domain has published a new schema document (version 4.1). Note the descendant relationships between the old and new data components. A schema document set consisting of Screening 1.1 and Justice 4.1 is incoherent because it refers to the old Justice 4.0 schema document outside the set, and therefore, violates the first criterion (the set must be closed). To [resolve] this we could incorporate the older 4.0 version into this set. Figure 2-2, *Incoherent schema set - incompatible data components*, below, indicates that adding Justice 4.0 violates the second criterion because multiple versions of the same component will exist that are incompatible. To make a coherent schema document set, either the Screening domain must be adjusted to use the new Justice 4.1 component or the schema document set must be revised to use the Screening domain with Justice 4.0 and not Justice 4.1.

Figure 2-1: Incoherent schema set - not closed

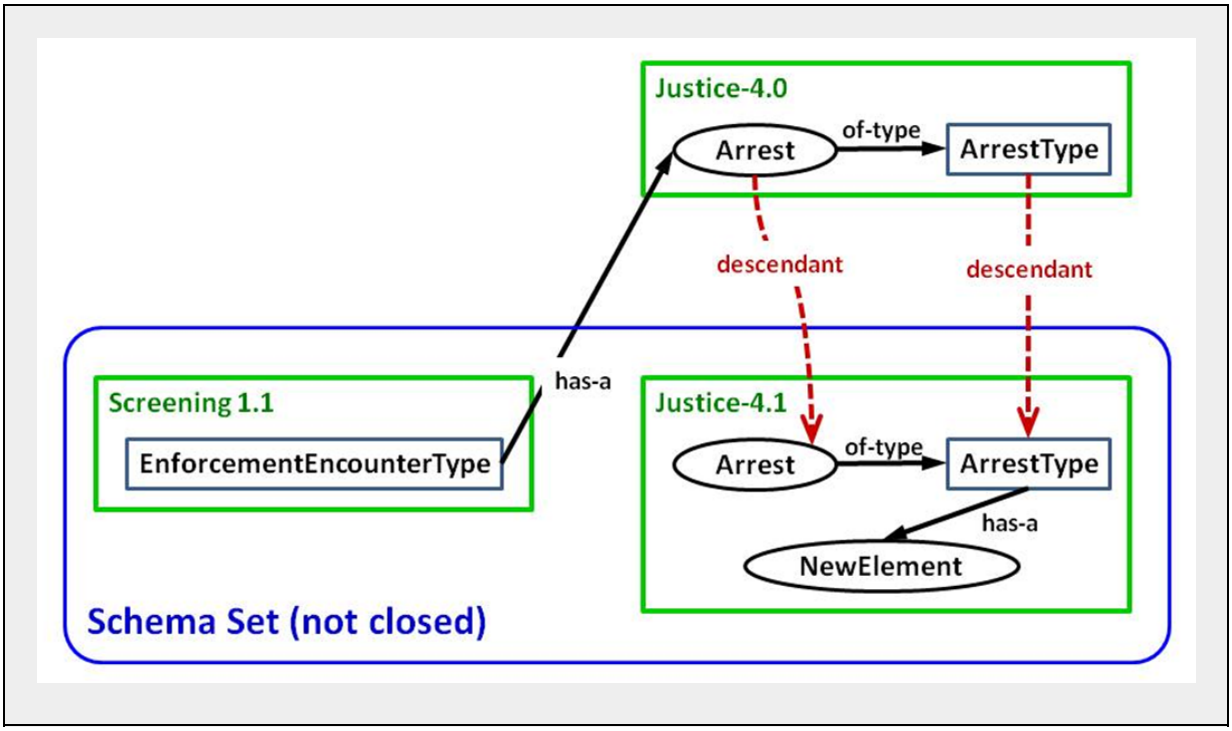
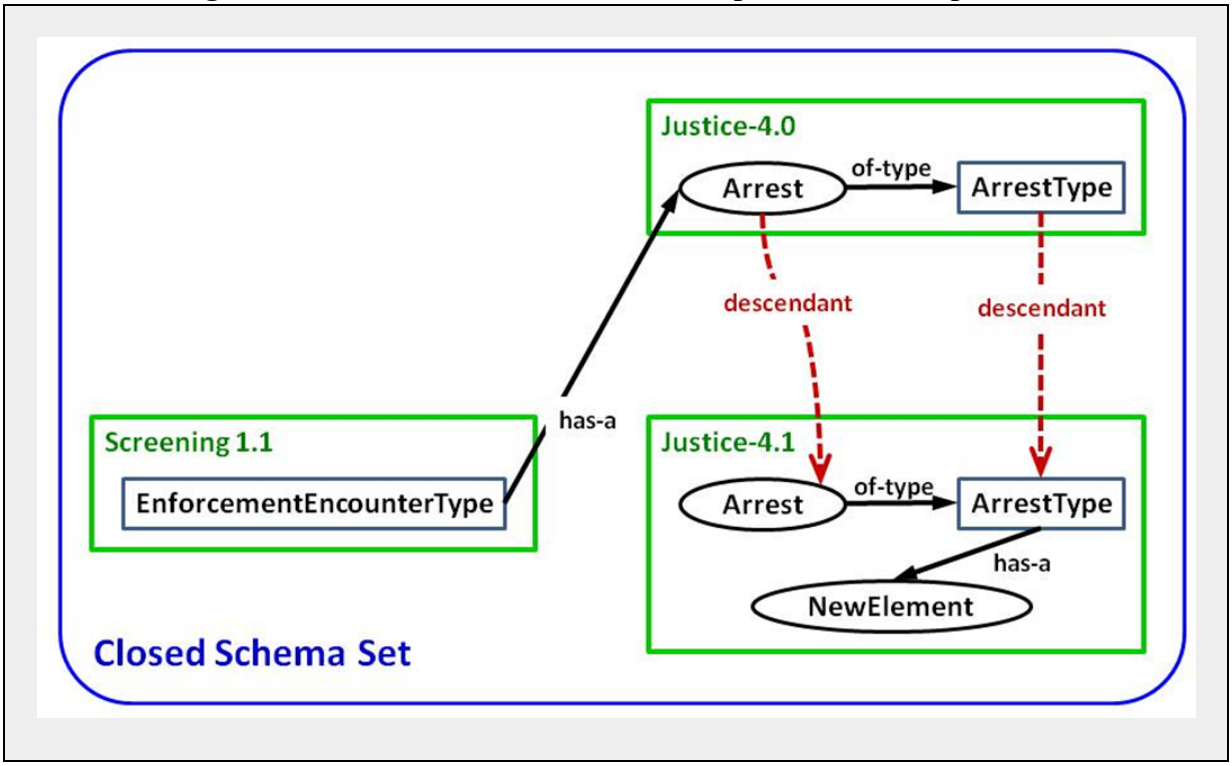


Figure 2-2: Incoherent schema set - incompatible data components



In general, two or more versions of a data component are incompatible when a type or element in one version of a schema has been copied to or redefined/redeclared in another, and both versions must exist in the same set because of cross referencing (as in the figure above). Note that even if all data components have not changed within two versions of the same schema document, a set that contains both schema documents will still be incoherent because the mere duplication of a data component in a new namespace is considered redefinition (and, of course, duplication).

However, two versions of a data component can also exist in a compatible way. The compatibility of two different versions of a data component depends on the way the ancestor component was changed to obtain the descendant. In Figure 2-2, *Incoherent schema set - incompatible data components*, above, Justice 4.1 and 4.0 `Arrest` elements are incompatible because the 4.1 version of `Arrest` was simply given an additional property (`NewElement`) and is essentially a redeclaration of the 4.0 version. This results in two semantically identical elements. In fact, as already mentioned, even if the `ArrestType` had remained the same across both versions, the 4.1 version is considered a redefinition and duplication of the 4.0 version.

On the other hand, if the 4.1 `ArrestType` had been derived (through type derivation) from the 4.0 version, and the 4.1 `Arrest element` had been made substitutable for the 4.0 version, then these components would be compatible. The difference is that these components have a clear relationship to their ancestors that is defined through XML mechanisms, whereas the former components do not. Furthermore, the substitutability property makes these components easily usable together (i.e., compatible).

The need to be a coherent schema document set is only required by official NIEM releases (major, minor, and micro). A core update is not absolutely required to be coherent with the core it applies to. However, except in rare cases, it will be crafted to be coherent. In order to provide flexibility to domains, a domain update schema document set is not required to be coherent. Whether or not a domain update is coherent with a given release depends on the content changes it applies as recorded in its change log.

2.9. MPD Types

This section details the five classes of MPDs currently defined in NIEM.

2.9.1. NIEM Release

A NIEM *release* is an MPD containing a full set of harmonized reference schema documents that coherently define and declare all content within a single version of NIEM. NIEM releases include major, minor, and micro releases (as defined in the **[NIEM High-Level Version Architecture]**).

[Definition: release]

A reference schema document set published by the NIEM Program Management Office (PMO) at `http://release.niem.gov/` and assigned a unique version number. Each schema document in the set defines data components for use in NIEM information exchanges. Each release is independent of other releases, although a schema document may occur in multiple releases. A release is of high quality, and has been vetted by NIEM governance bodies. A numbered release may be a major, minor, or micro release.

Current real examples of NIEM releases include NIEM major releases 1.0, 2.0, and 3.0, and minor release 2.1. Each numbered release is a reference schema document set that includes a NIEM Core (along with the various infrastructure and code list schema documents that supplement Core) and NIEM domain schema documents.

[Definition: major release]

A NIEM release in which the NIEM Core reference schema document has changed since previous releases. The first integer of the version number indicates the major release series; for example, versions 1.0, 2.0, and 3.0 are different major releases.

[Definition: minor release]

A NIEM release in which the NIEM Core has not changed from previous releases in the series, but at least one or more domain reference schema documents have changed. A second digit greater than zero in the version number indicates a minor release (for example, v2.1). Note also that major v2.0 and minor v2.1 are in the same series (i.e., series 2) and contain the same NIEM Core schema document.

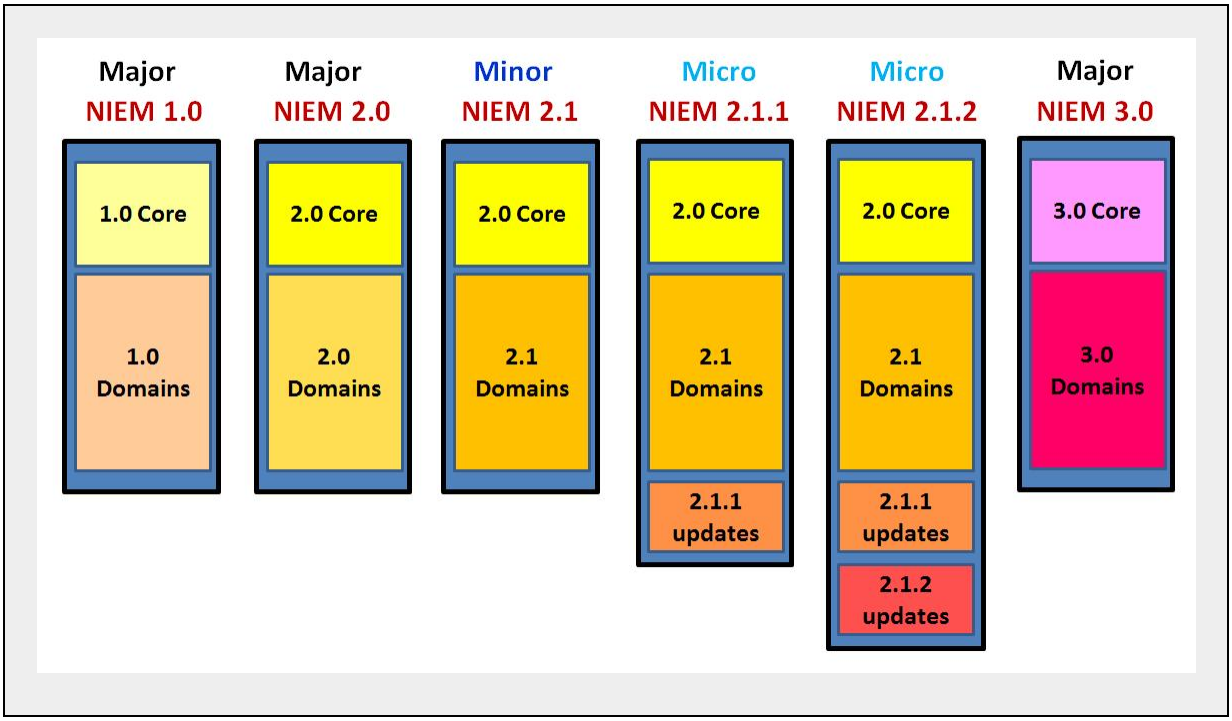
[Definition: micro release]

A NIEM release in which neither the NIEM Core nor the domain reference schema documents have changed from the previous major or minor release, but one or more new reference schema documents have been added (without impact to domain or Core schemas). A third digit greater than zero in the version number indicates a micro release (for example, v2.1.1 note that this release does not exist as of this date).

A micro release is a NIEM release that adds new data components to the Core, domains, or both without removing or modifying existing Core and domain schemas or content. Figure 2-3, *Examples of NIEM numbered releases*, below, illustrates both real (v1.0, v2.0, v2.1, and v3.0) and fictitious (v2.1.1 and v2.1.2) examples of major, minor, and micro release composition.

Note that a given NIEM reference schema document (target namespace) can exist in multiple numbered releases. For example, as illustrated in Figure 2-3, *Examples of NIEM numbered releases*, below, both NIEM 2.0 and 2.1 contain (and reuse) the same NIEM Core 2.0 schema document. Reuse of schema documents among releases is carefully coordinated to ensure coherence is maintained within each release. The **[NIEM High-Level Version Architecture]** defines the processes for numbering releases and identifying the schema documents that compose these sets. Later, this specification will outline a similar version numbering scheme for MPDs and their artifacts.

Figure 2-3: Examples of NIEM numbered releases



2.9.2. Domain Update

A *domain update (DU)* is an MPD containing a reference schema document or document set and a change log that represent changes to NIEM domains. The **[NIEM High-Level Version Architecture]** defines a domain update as both a process and a NIEM product. Through use and analysis of NIEM releases and published content, domain users will identify issues and new data requirements for domains and sometimes Core. NIEM domains use these issues as the basis for incremental improvements, extensions, and proposed changes to future NIEM releases. Both the process and product of the process are referred to as domain update. This MPD Specification is applicable to a domain update product.

[Definition: domain update]

An MPD that contains a reference schema document or document set issued by one or more domains that constitutes new content or an update to content that was previously published in a NIEM release. Domain updates are published to the NIEM Publication Area at `http://publication.niem.gov/niem/` and available for immediate use within IEPDs.

A domain update may define and declare new versions of content applicable to a NIEM release or other published content. The issuing domain or domains vet each update, but the update is not subject to review by other NIEM governance. Before publication, domain updates are technically reviewed for and must satisfy NIEM-conformance, but otherwise have fewer constraints on quality than do NIEM releases.

A domain update may apply to one or more domain namespaces within a single NIEM major, minor, or micro release. A domain steward uses a domain update to: (1) make new or changed domain content immediately available to NIEM data exchange developers between NIEM releases, and (2)

request that new or changed content be harmonized into a future NIEM release. (See **[NIEM Domain Update Specification]** which provides normative details about domain updates and the associated processes.)

2.9.3. Core Update

When necessary, the NIEM PMO can publish a *core update (CU)*. This is essentially identical to a domain update in terms of structure and use, with two important exceptions. First, a core update records changes that apply to a particular NIEM core version or another core update. This also means it is applicable to all NIEM releases using that same core version. Second, a core update is never published to replace a NIEM core. It is intended to add new schemas, new data components, new code values, etc. to a core without waiting for the next major release. In some cases, minor modifications to existing data components are possible.

> **[Definition: core update]**
>
> An MPD that applies changes to a given NIEM core schema document or document set. A core update never replaces a NIEM core; instead, it is used to add new schema documents, new data components, new code values, etc. to a particular NIEM core. In some cases, a core update can make minor modifications to existing core data components.

As with domain updates, all core updates are published to the NIEM Publications Area, their changes are immediately available for use in IEPDs, and they will be harmonized and integrated into the next major NIEM release.

2.9.4. Information Exchange Package Documentation (IEPD)

NIEM *Information Exchange Package Documentation (IEPD)* is an MPD that defines a class of instance XML documents that represent a recurring XML data exchange.

> **[Definition: Information Exchange Package Documentation (IEPD)]**
>
> An MPD that defines one or more (generally recurring) XML data or information exchanges.

A NIEM IEPD is a NIEM-conformant XML schema document set that may include portions of a NIEM Core schema document (and updates), portions of NIEM Domain schema documents (and updates), enterprise-specific or IEPD-specific extension schema documents, and declares at least one [IEP conformance target] within its MPD Catalog Section 4.1, *NIEM MPD Catalog*, below. The XML schema documents contained in an IEPD work together to define one or more classes of instance XML documents that consistently encapsulate data for meaningful information exchanges. Furthermore, any instance XML document that is valid for an XML schema document set (in the IEPD) and an associated [IEP conformance target] (declared in the IEPD) is considered a member of that [IEP conformance target] class. XML schema documents in a NIEM IEPD conform to the **[NIEM NDR]** and may use or extend data component definitions drawn from NIEM. An IEPD may

also incorporate and use XML schema documents from other standards that do not conform to NIEM. (See **[NIEM NDR]** for details.)

An IEPD consists of a set of artifacts (XML schema documents, documentation, sample instance XML documents, etc.) that together define and describe an implementable NIEM information exchange. An IEPD should contain an XML schema document set and instructional material necessary to:

- Understand information exchange context, content, semantics, and structure.
- Create and validate XML documents defined by the IEPD, and used for information exchanges.
- Identify the lineage of the IEPD itself and optionally its artifacts.

A NIEM IEPD defines one or more classes of XML documents. Each of these XML documents is an *Information Exchange Package (IEP)* that satisfies all validity constraints for its class as defined by the IEPD. An IEP is an information message payload serialized as XML and transmitted in some way, for example over a communications network. (**[FEA Data Reference Model]** and **[GJXDM IEPD Guidelines]** are the original sources of the terms *information exchange package* and *information exchange package documentation*, respectively).

> **[Definition: Information Exchange Package (IEP)]**
>
> An XML document that satisfies all the validity constraints for its class as defined by a NIEM IEPD.

How to declare validity constraints for one or more IEP classes within an IEPD will be covered in more depth in Section 4.6, *Information Exchange Packages*, below.

Note that NIEM conformance does not require that an IEP be native XML on the transmission medium. A NIEM-conformant IEP may be encrypted, compressed (e.g., using **[PKZIP]**, **[RAR]**, **[W3-EXI]**, etc.), or wrapped within an envelope mechanism, as long as its original native XML form can be retrieved by the receiver.

2.9.5. Enterprise Information Exchange Model (EIEM)

As an organization develops IEPDs, it may realize that many of its IEPDs have similar business content. A collection of closely related business data could be organized at an object level and defined as extension data components. In NIEM, these extension components are referred to as *Business Information Exchange Components (BIECs)*, because they are either specific to an organization's business or they represent a more general line of business that crosses organizational lines. Often they are business data components developed and used by multiple organizations within the same community of interest. So, instead of an *organization*, it is more appropriate and provides better context if we use the term *information sharing enterprise*.

> **[Definition: Information Sharing Enterprise]**
>
> A group of organizations with business interactions that agree to exchange information, often using multiple types of information exchanges. The member organizations have similar

business definitions for objects used in an information exchange and can usually agree on their common BIEC names and definitions.

Information sharing enterprises may cross various levels of government and involve multiple business domains. They may be self-defining and can be formal (with specific governance) or informal and *ad hoc*. An information sharing enterprise is the primary entity that supports the development and management of BIECs and an associated Enterprise Information Exchange Model (EIEM) (to be discussed next). Henceforth, unless otherwise stated, all references to an enterprise will implicitly mean information sharing enterprise.

A *Business Information Exchange Component (BIEC)* **[NIEM BIEC]** is a NIEM-conformant content model in XML Schema for a data component that meets the specific business needs of an information sharing enterprise for exchanging data about something that is a part of one or more information exchanges. This data component is tailored and intended to be used consistently across multiple IEPDs built by an enterprise. A BIEC is a NIEM-conformant data component that is:

- Reused from a NIEM release (for example, as a subset; with possibly modified cardinality), or
- Extended per the **[NIEM NDR]** from an existing NIEM data component, or
- Created per the **[NIEM NDR]** as a new data component that does not duplicate existing NIEM components within a release in use.

[Definition: Business Information Exchange Component (BIEC)]

A NIEM-conformant XML schema data component definition or declaration (for a type, element, attribute, or other XML construct) reused, subsetted, extended, and/or created from NIEM that meets a particular recurring business requirement for an information sharing enterprise.

The use of BIECs has the potential for simplifying IEPD development and increasing consistency of the business object definitions at all steps in the process, including exchange content modeling, mapping to NIEM, creating NIEM extension components, and generating XML schema documents.

An *Enterprise Information Exchange Model (EIEM)* is an MPD that incorporates BIECs that meet enterprise business needs for exchanging data using **[NIEM BIEC]**s. An EIEM is an adaptation of NIEM schema documents, tailored and constrained for and by an enterprise. An EIEM contains the following schema documents that are commonly used or expected to be used by the authoring enterprise:

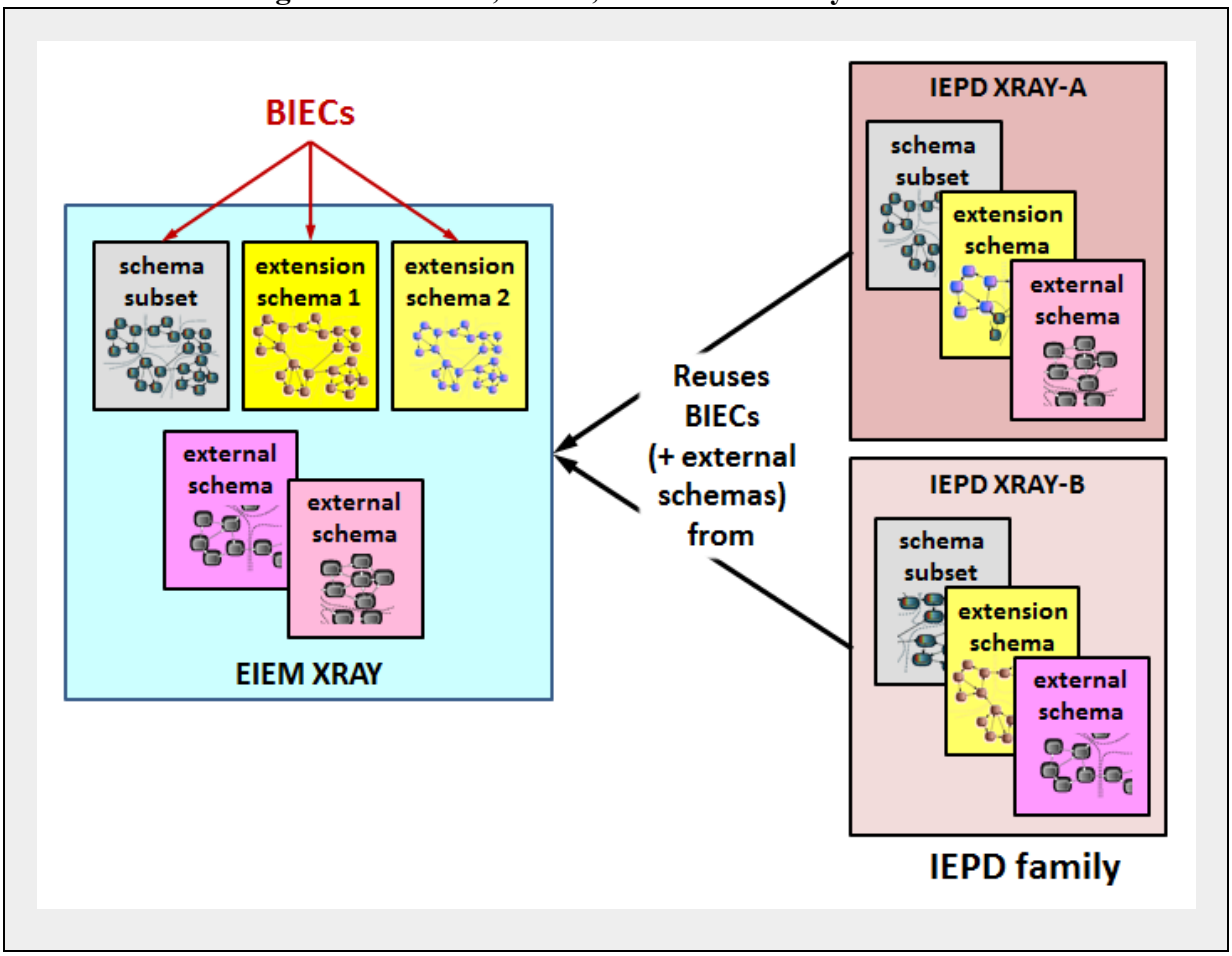
- One standard NIEM schema document subset (or reference schema document set).
- One or more NIEM extension schema documents that extend existing NIEM data components or establish new NIEM-conformant data components.
- Optionally, as needed, one or more NIEM constraint schema document sets (usually based on a schema document subset).
- Optionally, as needed, one or more XML schema documents for non-NIEM (i.e., non-conformant) standards with associated extension schema documents that contain adapter types for the data components that will be used from those non-NIEM XML schema documents (per

[NIEM NDR]).

> **[Definition: Enterprise Information Exchange Model (EIEM)]**
>
> An MPD that contains a NIEM-conformant schema document set that defines and declares data components to be consistently reused in the IEPDs of an enterprise. An EIEM is a collection of BIECs organized into a schema document subset and one or more extension schema documents. Constraint schema documents and non-NIEM-conformant external standards schema documents with type adapters are optional.

An information sharing enterprise that creates and maintains an EIEM authors IEPDs by reusing its EIEM content instead of (re)subsetting reference schema documents sets and (re)creating extensions. An EIEM may also contain business rules or constraint schema document sets tailored to enterprise requirements and designed to restrict variability in use of NIEM data components. This not only saves time, but it also ensures that enterprise IEPDs reuse NIEM and associated extensions consistently. (XML schema document subsets, extension schema documents, and constraint schema document sets will be defined and discussed in more detail later in this document). Figure 2-4, *BIECs, EIEM, and a small family of IEPDs.*, below, generally illustrates how BIECs, an EIEM, and an IEPD family relate (Constraint schema document sets are optional and not depicted in this figure).

Figure 2-4: BIECs, EIEM, and a small family of IEPDs.



2.10. Similarities and Differences of MPD Classes

It will be helpful to summarize the foregoing discussions by listing the primary similarities and differences among the various types of MPDs. This will help highlight the nature of this specification as a baseline and point of leverage for all five classes of MPDs: NIEM release, core update (CU), domain update (DU), IEPD, and EIEM. Note that these lists are not all inclusive.

MPD class similarities:

- Principal artifacts are XML schema documents (XSD), the purpose for which is to define and declare reusable data components for information exchanges or to define the exchanges themselves.
- Each MPD requires a self-documenting `mpd-catalog.xml` artifact containing metadata and a listing of its key artifacts. This establishes its name, version, class, purpose, general content, lineage, etc.
- Each MPD requires a change log.
- Each MPD requires a Uniform Resource Identifier (URI) and a version number.
- Each MPD must be packaged as a self-contained ZIP archive (in one form). Self-contained simply means that an MPD has copies of (not just URLs or references to) all schema documents needed to validate instance XML documents it defines.
- Each MPD may contain optional alternate representations besides XML Schema (for example, generic diagram, UML diagram, XMI, database format, spreadsheet, etc.).

MPD class differences:

- IEPDs and EIEMs contain subset, extension, external, and constraint schema documents and document sets. NIEM releases, core updates, and domain updates contain reference schema document sets.
- An IEPD must declare at least one [IEP conformance target] within its MPD Catalog Section 4.1, *NIEM MPD Catalog*, below. Other MPD classes do not have this requirement.
- EIEMs and domain updates may optionally contain sample instance XML documents and associated XSLT files to display them. NIEM releases and core updates do not.
- A domain update may supersede and replace another published schema document/namespace. It may also add to or modify content in another published schema document/namespace without including the unchanged content. Core updates may only add to (supplement); never a replacement for and never modifies a NIEM Core.
- IEPDs, EIEMs, and NIEM releases are independently complete. A core update can be issued as a new complete standalone reference schema document to be used with a NIEM core.

Table 2-1, *Comparison of MPD classes*, below, summarizes the similarities and differences of MPD classes by indicating the characteristics for each:

Table 2-1: Comparison of MPD classes

Characteristics of MPD Classes	Release	CU	DU	IEPD	EIEM
Requires a URI	X	X	X	X	X
Requires a version number	X	X	X	X	X
Must be packaged as a **[PKZIP]** archive	X	X	X	X	X
May contain alternate model representations (in addition to XSD)	X	X	X	X	X
Requires an `mpd-catalog.xml` artifact (specified by XSD)	X	X	X	X	X
Requires a formal XML change log (specified by XSD)	X	X	X		
Requires a change log but may be informal; any format				X	X

Requires a master document				X	X
Its XML schema document set defines reusable data components	X	X	X		X
Its XML schema document set defines data exchanges (IEPs)				X	
Can contain subset, extension, external, or constraint schema documents				X	
Contains subset, extension, or external schema documents; optionally constraint schema document sets				X	X
Contains reference schema documents only	X	X	X		
Must declare at least one or more [IEP conformance targets]				X	
May contain sample instance XML documents that validate to XML schema document set			X	X	X
Required to be independently complete standalone XML schema document set	X			X	X
May be independently standalone XML schema document set		X	X		
May supersede other published XML schema documents (target namespaces)			X		

3. MPD XML Schema Document Artifacts

XML schema document artifacts are the essential content of MPDs because they normatively define and declare data components. The purpose of an MPD is determined by the XML schema document or document set(s) it contains; furthermore, each schema document may have a different purpose. The **[NIEM NDR]** addresses some schema documents as conformance targets including reference schema documents, extension schema documents, and schema document sets. Each conformance target may adhere to a different (though possibly overlapping) set of conformance rules. Consult the **[NIEM NDR]** for these rules. NIEM also employs a special technique that relies on constraint schema documents and document sets.

The following subsections will define each type of NIEM schema document and document set, and will identify the types of MPDs that may or must contain them. The last subsection discusses sample instance XML documents (IEPs) that validate with IEPD schema document sets, and when such instance XML documents are mandatory.

3.1. Reference Schema Documents

This section generally applies to NIEM releases, core updates, and domain updates. Though not common, it is also valid to use a reference schema document or document set within an IEPD or EIEM. Reference schema document and reference schema document set were defined earlier in Section 2.7, *Reference Schema Documents*, above.

A NIEM reference schema document is intended to be the authoritative definition schema document for a NIEM target namespace, therefore, all NIEM releases, core updates, and domain updates are composed of a reference schema document set and associated namespaces. As a standalone artifact, a reference schema document set is always coherent and harmonized such that all types and properties are semantically unique (i.e., multiple versions of semantically identical types or properties do not exist within the set).

As authoritative definitions, NIEM reference schema document sets satisfy more rigorous documentation requirements. The **[NIEM NDR]** requires that each type definition, and element and attribute declaration in a reference schema document contain an `xs:annotation` element that defines its semantic meaning. As will be explained later, extension schema documents are also authoritative definitions, but in a local sense. They are authoritative within a given IEPD or EIEM, and therefore, must also satisfy the same rigorous documentation rules as reference schema documents.

Typically reference schema documents contain data components with the most relaxed cardinality (`0` to `unbounded`). However, this is not an absolute requirement. Cardinality in reference schema documents may be constrained if necessary to model reality. For example, one might claim that NIEM releases should restrict `PersonType` to a single occurrence of the element `PersonBirthDate`. Every person has one and only one birth date. Unfortunately, also in reality, criminal persons often present multiple identities with multiple birth dates; and so the capability to represent such is an important data requirement for NIEM.

3.2. Subset Document Schemas

This section only applies to IEPDs and EIEMs. NIEM releases, core updates, and domain updates do not contain schema document subsets (only reference schema document sets).

3.2.1. Basic Subset Concepts

A NIEM *schema document subset* is a set of XML schema documents that constitutes a reduced set of components derived from a NIEM reference schema document or document set associated with a given numbered release or domain update. Any given XML schema document within a schema document subset is referred to as a *subset schema document* (terms reversed).

[Definition: subset schema document]

An XML schema document that meets all of the following criteria:

- It is built from a reference schema document set where one or more reference schema documents has been substituted by a its corresponding subset schema document.
- It is built from a reference schema document by applying subset operations to the XML schema statements in a reference schema document.
- It is explicitly designated as a subset schema document. This is accomplished by declaration in the relevant MPD catalog or by a tool-specific mechanism outside the subset schema document.
- It has a target namespace previously defined by a reference schema document. That is, it does not provide original definitions and declarations for schema components, but instead provides an alternate schema representation of components that are defined by a reference schema document.
- It does not alter the business semantics of components in its namespace. The reference schema document defines these business semantics.
- It is intended to express the limited vocabulary necessary for an IEPD or EIEM and to support XML Schema validation for an IEPD.

See also [schema document subset].

The primary purpose for a schema document subset is to reduce and constrain the scope and size of a full NIEM reference schema document set for use within an IEPD or EIEM. Thus, a schema document subset is derived from a reference schema document set (such as a NIEM release) by applying subset operations (See Section 3.2.2, *Subset Operations*, below). Also, note that the process of deriving a schema document subset from a NIEM reference schema document set is optional; it is completely valid to reuse NIEM reference schema documents as-is within IEPDs or EIEMs.

[Definition: schema document subset]

An XML schema document set built from a reference schema document set by applying subset operations to the reference schema documents in that set. See also [subset schema document].

Because NIEM adopts an optional and over-inclusive data representation strategy, most elements in a NIEM reference schema have zero to unbounded cardinality. So, elements with cardinality `minOccurs="0"` are optional and may be omitted from a subset schema document if not needed for business reasons. It is also valid to constrain element cardinality within a subset schema document, as long as doing so does not break the subset relationship with the reference schema document set. For example, a reference schema document element with cardinality (`minOccurs="0"`, `maxOccurs="unbounded"`) may be constrained to (`0`,`1`) or (`1`,`1`) in a subset schema document. However, if a reference schema document element's cardinality is (`1`,`unbounded`), it may not be constrained to (`0`,`1`) since this breaks the subset relationship. The interval (`0`,`1`) is not contained within, and instead, overlaps the interval (`1`,`unbounded`).

The fundamental rule for a valid schema document subset is as follows:

[Rule 3-1]

Any instance XML document that validates against a NIEM schema document subset will validate against the NIEM reference schema document set from which that schema document subset was derived.

3.2.2. Subset Operations

NIEM subset operations are essentially reduction operations that remove or constrain portions of a reference schema document set, thereby building a profile of the set. They do not expand the scope (i.e., relax constraints) or change the semantics of reference schema document set content.

The following describe valid operations that will produce a [schema document subset]:

1. Remove an XML comment statement.
2. Remove an `xs:annotation` (includes `xs:documentation` and `xs:appinfo`).
3. Increase the value of an `xs:element/@minOccurs` attribute (must be less than or equal to

`maxOccurs` value).

4. Decrease the value of an `xs:element/@maxOccurs` attribute (must be greater than or equal to `@minOccurs` value).
5. Remove an `xs:element` if `@minOccurs="0"`.
6. Remove an `xs:complexType` or `xs:simpleType` (if not supporting an element or attribute).
7. Remove an `xs:attribute` (if `@use="optional"`) from an `xs:complexType`.
8. Change an `xs:attribute/@use="optional"` to `@use="prohibited"`.
9. Change an `xs:attribute/@use="optional"` to `@use="required"`.
10. Remove an `xs:schema/xs:element` declaration (if not supporting an element use).
11. Remove an `xs:enumeration` from an `xs:simpleType` (unless it is the only remaining `xs:enumeration`).
12. Add or apply a constraining facet to an `xs:simpleType`.
13. Remove an `xs:import` and its associated schema document (if the schema document is not used within the document set).
14. Change a concrete `xs:/schema/xs:element` declaration to `@abstract="true"`.
15. Change an element from `nillable="true"` to `nillable="false"`.
16. Substitute an `xs:element/@substitutionGroup` member for its associated substitution head.
17. Substitute a composition of `xs:element/@substitutionGroup` members for their associated substitution head (subject to cardinality and unique particle attribution (UPA) constraints). The composition is an ordered sequence of the `@substitutionGroup` member elements. Each substitute element may bound its cardinality such that the total cardinality sum is within the bounds of the `@substitutionGroup` head cardinality. Order and cardinality of the replacement sequence must conform to XML Schema UPA constraints.
18. Replace a wildcard (subject to cardinality, UPA, and namespace constraints) with a composition, i.e., an ordered sequence of elements. Each element may further bound cardinality within the bounds of the wildcard. Order and cardinality of replacement sequence must conform to XML Schema UPA constraints. The namespace of each element must conform with namespace constraints specified by the wildcard (if any).

3.2.3. Subset Schema Document Namespaces

A schema document subset is essentially a reference schema document set (i.e., a numbered release) that has been modified by applying the foregoing subset operations to support business requirements represented in an IEPD or EIEM. A subset derived from a reference schema document set may differ from that reference set only in that its content has been reduced and/or constrained. For this reason, each subset schema document adopts the target namespace of its corresponding reference schema document.

> **[Rule 3-2]**
>
> Each subset schema document in a schema document subset derived from a reference schema document set bears the same target namespace as the schema in the reference schema document set on which it is based.

3.2.4. Multiple Schema Document Subsets in a Single IEPD or EIEM

This section only applies to NIEM IEPDs and EIEMs. NIEM releases, core updates, and domain updates do not contain schema subsets.

Previous sections defined a single schema subset derived from a reference schema document set. In general, an IEPD or EIEM contains a single cohesive schema subset (which may be a rather large set of files) based on one numbered NIEM release or domain update.

However, this specification does not restrict the number of different subsets that may be employed within a single IEPD or EIEM. Furthermore, it does not restrict the employment of subsets from different numbered releases within a single IEPD or EIEM. However, exercising this degree of flexibility makes it critically important that developers understand the potential consequences. NIEM subsets represent a delicate compromise between flexibility and interoperability. On the one hand, a set of IEPDs based on the same subset and numbered release use identical data components, thereby enhancing interoperability. On the other hand, mixing dissimilar subsets from the same numbered release or mixing subsets derived from various numbered releases has the potential to negatively impact interoperability through incoherence and ambiguity.

The NIEM mandate that every schema have a unique target namespace prevents name conflicts between reference schema document sets and between two subsets derived from different reference sets. In spite of namespace distinction, mixing subsets of multiple reference schema document sets can still introduce multiple versions of semantically equivalent data components, a potentially ambiguous situation. Even employing multiple subsets together that have been derived from the same reference set has the potential to create a similar result. Above all, it is the developer's responsibility to ensure that, if mixing subsets from one or more numbered releases within a single IEPD or EIEM, these artifacts are carefully coordinated and clearly documented to ensure the various versions of semantically equivalent data components and different schemas with the same namespaces will not cause conflicts, confusion, and/or failure during validation or exchange implementation.

3.3. Extension Schema Documents

This section only applies to NIEM IEPDs and EIEMs. NIEM releases, core updates, and domain updates do not contain extension schema documents.

[Definition: extension schema document]

A NIEM-conformant schema document that adds domain or application specific content to the base NIEM model.

The **[NIEM NDR]** defines an IEPD *extension schema document* as a conformance target. In general, an extension schema document contains components that use or are derived from the components in reference schema documents. It is intended to express the additional vocabulary required for an IEPD, above and beyond the vocabulary available from reference schema documents.

An IEPD or EIEM developer who determines that NIEM is missing elements required for a given information exchange has three options to account for such requirement shortfalls. Using rules and techniques outlined in the **[NIEM NDR]**:

- Extend an existing NIEM data component (if possible).

- Augment an existing NIEM data type (through NIEM Type Augmentation).
- Build a new NIEM-conformant data component.

A NIEM extension schema document may contain data components built from both options above. Employment of extension schema documents in an IEPD is entirely optional.

Multiple extension schema documents are allowed in a single IEPD. Developers will likely want to reuse many of their extension schema documents in other IEPDs. Therefore, the best practice for extension is to group all data components designed to reuse into one extension schema document or document set, and group IEPD-specific data components into another. Then the reusable extension components can be more easily redeployed in other IEPDs as needed. Also, recall that Section 2.9.5, *Enterprise Information Exchange Model (EIEM)*, above, discusses EIEM employment for larger scale reuse of NIEM data components in multiple IEPDs.

Extension schema documents generally contain new data component declarations that may (though not necessarily) be derived from or reference existing NIEM data components. This being the case, reference schema documents do not exist for new data components found within extension schema documents. Therefore, extension schema documents must satisfy the more rigorous documentation requirements of reference schema documents. Per the **[NIEM NDR]**, the definition or declaration of each new data component in an extension schema document must include an `xs:annotation` element that provides its semantics and NIEM-specific relationships.

3.4. External Schema Documents

NIEM allows the use of *external schema documents* that do not conform to NIEM. Data components declared and defined in external schema documents require NIEM *adapter types* to identify the fact they do not conform to NIEM.

> **[Definition: external schema documents]**
>
> Any XML schema document that is not a NIEM-supporting schema and that is not NIEM-conformant.

Refer to the **[NIEM NDR]** for details about external schemas, adapter types, and the rules describing their usage.

3.5. Constraint Schema Documents and Document Sets

This section only applies to NIEM IEPDs and EIEMs that may use constraint schema documents or document sets. NIEM releases, core updates, and domain updates do not contain constraint schema documents.

A *constraint schema document* is an optional IEPD or EIEM artifact that is used to express business rules for a class of instance XML documents, and is not assumed to be a definition for the semantics of the components it contains and describes. Instead, a constraint schema document uses the XML Schema Definition Language to add constraints to components defined or declared by other schema documents, usually from a schema document subset.

[Definition: constraint schema document]

A schema document that imposes additional constraints on NIEM-conformant instance XML documents. A constraint schema document or document set validates additional constraints imposed on an instance XML document only after it is known to be NIEM-conformant (i.e., has been validated with a reference schema document set, or schema document subset, and applicable extension schema documents). Constraint schema document validation is a second-pass validation process that occurs independently of and after conformance validation. A constraint schema document need not validate constraints that are applied by other schema documents. See also [constraint schema document set].

[Definition: constraint schema document set]

A set of related constraint schema documents that work together, such as a constraint schema document set built by adding constraints to a schema document subset. See also [constraint schema document].

Constraint schema documents are generally useful when it is necessary to impose restrictions that are more complex than cardinality. If only cardinality restrictions are needed, then it is easier and more efficient to set these directly in the subset schema documents and avoid the use of constraint schema documents. Otherwise, constraint schema documents may be necessary. Note however, that any cardinality restrictions placed on NIEM release components within schema document subsets must not violate the rules established in Section 3.2.1, *Basic Subset Concepts*, above, which define the relationship of the reference schema document to a subset schema document derived from it.

The **[NIEM NDR]** provides a normative definition and description of constraint schema documents. However, a few points are worth mentioning here.

Use of constraint schemas is one option for applying additional business rules to or tightening constraints on NIEM IEPs beyond what NIEM itself provides. This particular technique uses the XML Schema Definition Language **[W3C XML Schema Datatypes]**, **[W3C XML Schema Structures]**. NIEM also allows other methods that do not use XML Schema, such as **[ISO Schematron]** or other methods. However, at this time there are no normative rules for how these techniques should be employed in NIEM IEPDs or EIEMs. Therefore, if other techniques are used, it is a developer responsibility to incorporate appropriate artifacts and clear documentation.

Constraint schema documents are generally designed and employed in sets, similar to reference schema document set or schema document subsets. A common practice for creating an IEPD or EIEM constraint schema document set is to start with a valid NIEM schema document subset and modify it to further restrict the class of instance XML documents (IEPs) that will validate with this constraint schema set. However, an extension schema document can also be used to derive a constraint schema document. The namespace of a constraint schema document is established the same way the namespace of a subset schema document is established, by reusing the target namespace of the schema document from which it is derived.

[Rule 3-3]

A constraint schema document MUST bear a target namespace that has been previously assigned to a reference or extension schema document, or is a constraint schema document intended to support another constraint schema document that has such a target namespace.

To use a constraint schema document set to tighten constraints on IEPs, a two-pass validation technique is employed. In the first pass, an IEP is validated against the schema document subset and extension schema documents. This pass ensures that IEP semantics and structure conform to the NIEM model and NDR. In the second pass, an IEP is checked against a constraint schema document set, which may contain constrained versions of the subset schema documents and extension schema documents. This pass ensures that the IEP also satisfies the additional constraints (i.e., business rules that the first pass was unable to validate).

There is no restriction on the number of constraint schema document sets that an IEPD or EIEM can employ. As in other advanced situations, developers must clearly document their intentions for and use of multiple constraint schema document sets.

In general, constraint schema documents have far fewer requirements than other classes of NIEM schema documents. Since they work in tandem with NIEM normative schema documents, constraint schema documents are allowed to use the XML Schema Definition language in any way necessary to express business rules. This means that to constrain instance XML documents, constraint schema documents can employ XML Schema constructs that are not allowed in other classes of NIEM schema documents.

BIECs in particular may have additional business rules in constraint schema documents. A normative NIEM BIEC Specification (not available at the time of the publication of this MPD Specification), will supplement or obviate constraint schema documents with consistent and formal techniques for representing business rules within NIEM components. However, as already mentioned, the MPD Specification does not prohibit or restrict the application of formal business rule techniques (such as **[ISO Schematron]**) to MPDs now.

3.6. Classes of MPDs vs. Classes of Schema Documents

The chart in Table 3-1, *Schema document classes vs. MPD classes*, below, summarizes the types of schema documents that: (1) can be contained in an instance of each MPD class, and (2) the (*minimum, maximum*) cardinalities of those schema documents. In some cases, certain types of schema documents are never contained in particular class of MPD. These are labeled "*not applicable*".

Notice that only NIEM releases, core updates, and domain updates contain reference schema document sets, while only IEPDs and EIEMs contain the user-developed schema document sets. The pluses (+) indicate that a NIEM-conformant IEPD or EIEM must have at least one schema document that is either a NIEM reference schema document or a NIEM subset schema document derived from a NIEM reference schema document (See **[Rule 3-4]**, below).

Table 3-1: Schema document classes vs. MPD classes

Schema Document					

Classes	Release	CU	DU	IEPD	EIEM
Reference	(1, unbounded)	(1, unbounded)	(1, unbounded)	(0+, unbounded)	(0+, unbounded)
Subset	not applicable	not applicable	not applicable	(0+, unbounded)	(0+, unbounded)
Constraint	not applicable	not applicable	not applicable	(0, unbounded)	(0, unbounded)
Extension	not applicable	not applicable	not applicable	(0, unbounded)	(0, unbounded)
External	(0, unbounded)	(0, unbounded)	(0, unbounded)	(0, unbounded)	(0, unbounded)

[Rule 3-4]

A NIEM-conformant IEPD or EIEM MUST contain at least one schema document that is either a NIEM reference schema document or a subset schema document derived from a NIEM reference schema document.

4. MPD Documentation Artifacts

XML schema documents (and the schemas that result from them) are the essence of a NIEM MPD. All other artifacts are considered documentation.

A variety of documentation files may be incorporated into a NIEM MPD. However, in addition to XML schema documents, there are only two mandatory documentation artifacts required by every MPD: the *mpd-catalog* and the *change log*. An mpd-catalog (`mpd-catalog.xml`) contains basic metadata, relationship and lineage data, and validation information. The change log provides a history of modifications.

A *master document* is mandatory for IEPDs and EIEMs. These MPD classes may be built by different developers, and may be registered into a repository for reuse by many other users, developers, and implementers; therefore, a minimal form of documentation is absolutely necessary. An IEPD or EIEM master document is the primary source and starting point for human readable documentation (similar to a `readme` file), and should reference (and describe) any other separate documentation artifacts. This requirement ensures that baseline documentation is consistently rooted in a clearly visible artifact within each IEPD and EIEM.

The following subsections will address these artifacts and the concepts, metadata, and content each supports.

4.1. NIEM MPD Catalog

[Definition: MPD catalog document]

An XML document that is Schema valid to `mpd-catalog-3.0.xsd` in Appendix A, *MPD Catalog XML Schema Document*, below. The *MPD catalog document* contains metadata that describes:

- Unique identification
- Basic characteristics and properties
- Key artifacts and directory structure
- Relationships to other MPDs and their artifacts
- [IEP conformance targets]

All MPDs require an `mpd-catalog.xml` artifact. So, the [MPD catalog document] is tailored to accommodate all MPD classes. However, each MPD class has different catalog requirements. The catalog metadata are formally defined and declared in an XML schema Appendix A, *MPD Catalog XML Schema Document*, below.

This metadata is designed to be the minimal needed to facilitate human understanding, tool support, and machine processing. The metadata can support a number of MPD uses and functions including (but not limited to):

- Identification of key artifacts
- Generation of a hyperlinked content display using XSLT
- Browsing and understanding of artifacts and their content
- Automatic registration into a registry/repository
- Search, discovery, retrieval of MPDs (through metadata and relationships)
- Reuse of MPDs and their artifacts
- Reuse of BIECs and associated EIEMs
- Tracing and analysis of MPD lineage
- General conformance and validation of the MPD itself
- Definition, identification, and validation of IEP conformance targets

[Rule 4-1]

A [well-formed] and [complete MPD] MUST contain an mpd-catalog XML document artifact that:

- validates with the NIEM MPD catalog schema contained in Appendix A, *MPD Catalog XML Schema Document*, below.
- validates with the NIEM MPD catalog Schematron rules [TBD].
- resides in the [MPD root directory].
- bears the file name `mpd-catalog.xml`.

Note that the Appendix A, *MPD Catalog XML Schema Document*, below, is a fairly relaxed XML schema definition with very few mandatory properties. The reason for this is to support tools that must identify and distiguish a [well-formed] from a [complete MPD] during incremental stages of development. This is explained in more detail in Section 5, *MPD Resolution, Existence, and*

Validation Rules, below.

Because the baseline XML Schema definition for an MPD catalog is more relaxed than is required for a [complete MPD], special Schematron rules must also be applied to an MPD catalog to ensure its validity or completeness Section 5, *MPD Resolution, Existence, and Validation Rules*, below.

4.1.1. MPD Catalog as a Table of Contents

One function of the MPD catalog is to serve as a table of contents that identifies, locates, and classifies key artifacts and artifact sets. For that purposes Appendix A, *MPD Catalog XML Schema Document*, below, provides a number of classifier elements for most common artifacts in MPDs. For other less common or generic artifacts two general classifiers exist: `Documentation` and `ApplicationInfo`. These elements loosely correspond to the meaning of the XML Schema `xs:annotation` child elements, `xs:documentation` and `xs:appInfo`. General visual, audio, and textual explanatory documentation should be classified as `Documentation`, while tool-specific artifacts (such as imports, exports, executables, etc.) should be classified as `ApplicationInfo`.

The classifier elements are designed to identify, categorize, and describe any artifact (including its path name, dependencies, and lineage). Employing XSLT, `mpd-catalog.xml` can be transformed into an `index.html` artifact that displays a hyperlinked MPD table of contents and metadata summary in a browser.

In general, only an IEPD or EIEM would contain `Documentation` and `ApplicationInfo` artifacts. So, for an IEPD and EIEM, a best practice is to use the `master-document` artifact (i.e., the [master document] required in the [MPD root directory]) to reference `Documentation` and `ApplicationInfo` artifacts whether they have been classified in the mpd catalog or not.

Release, core update, and domain update catalogs are required to record all artifacts. However, IEPD and EIEM MPD catalogs are not. The IEPD or EIEM author decides which artifacts (both files and sets) are important enough to explicitly include in the MPD catalog. The author may choose to include all, some, or no artifacts in the catalog.

Also note that use of the MPD catalog as a table of contents, frees an IEPD or EIEM author from having to use a standard directory structure. Instead, the author can design his/her own or use guidance provided in Appendix D, *Guidance for IEPD Directories (non-normative)*, below.

4.1.2. Extending an MPD Catalog

An MPD Catalog may be extended to accommodate new or additional metadata, artifact classifiers, or validity constraints that are not already defined in Appendix A, *MPD Catalog XML Schema Document*, below.

To extend the mpd-catalog, an MPD author must provide both an XML catalog extension document (XML) and one or more MPD extension schema documents (XSD). The xml-catalog extension identifies that one or more MPD catalog extensions are present, and resolves their namespaces to local URIs. The MPD catalog extension is a schema that defines and declares the new data components for metadata, classifiers, and/or constraints. Both general **[NIEM Conformance]** and specific **[NIEM NDR]** NIEM conformance rules apply to these components. The xml-catalog extension document must reside in the [MPD root directory]. The MPD extension schema documents may bear any file name and reside anywhere in the MPD. This is because the xml-catalog is expected to [resolve] all

local URIs. MPD processing tools are expected to look for and recognize the xml-catalog (that identifies MPD catalog extensions exist) by its file name.

The following rule specifies the requirements for an mpd-catalog extension XML catalog document:

[Rule 4-2]

An MPD extension XML catalog document:

- MUST reside in the same relative directory as the `mpd-catalog.xml` artifact (normally in the [MPD root directory])
- MUST bear the file name (and type) `mpd-extension-xml-catalog.xml`.
- MUST [resolve] all MPD catalog schema extension document namespaces to the correct corresponding local URIs in the MPD.

So, when a processer identifies a file named `mpd-extension-xml-catalog.xml` in the [MPD root directory], it can assume that it contains references to one or more MPD catalog extension schema documents. These schema documents have the following requirements:

[Rule 4-3]

An MPD catalog extension schema document:

- MUST conform to the **[NIEM NDR]**, specifically the extension schema conformance target rules.
- MUST bear the file type of `.xsd`
- MAY have any file name.
- MAY reside anywhere within the MPD (the XML catalog MUST [resolve] its URI).

The new data components defined/declared within an MPD catalog extension schema document must adhere to the following rule:

[Rule 4-4]

An MPD-catalog-extension-schema-document-data-component:

- MUST conform to the **[NIEM Conformance]** Specification.
- MUST conform to the **[NIEM NDR]** (extension schema document conformance target).
- MUST employ XSD type derivation (`xs:extension` or `xs:restriction`) and/or element substitution.

Whether extending an MPD catalog with new metadata elements, artifact classifier elements, or validity constraint elements, Appendix A, *MPD Catalog XML Schema Document*, below, provides an abstract element as a substitution group head in each case. The user simply derives a new type

(through extension or restriction), or reuses an existing type, then declares a new element (of that type), and identifies it with the appropriate substitution group. Whenever possible, the user should reuse types, elements, and attributes that are already defined/declared within the Appendix A, *MPD Catalog XML Schema Document*, below.

4.2. Metadata Concepts

The mpd-catalog also contains both required and optional metadata for the MPD and its artifacts. The following subsections specify the syntax, formats, and semantics for that metadata.

4.2.1. Version Numbering Scheme

Published MPDs will be periodically revised and updated; therefore, versioning is required to clearly indicate changes have occurred. A version number is actually part of the unique identification for an MPD (to be discussed in Section 4.2.2, *URI Scheme for MPDs*, below).

In order to maintain some consistency while allowing reasonable flexibility to authors, this specification establishes a simple version numbering scheme that is consistent with most common practices. This is the same version numbering scheme that is used for NIEM releases.

[Rule 4-5]

Every MPD MUST be assigned a version number that adheres to the regular expression:

```
version ::= digit+ ('.' digit+)* (status digit+)?
Where:
        digit   ::= [0-9]
        status  ::= 'alpha' | 'beta' | 'rc' | 'rev'
```

The meaning of `status` value options are as follows:

- `alpha` indicates early development; changing significantly.
- `beta` indicates late development; but changing or incomplete.
- `rc` indicates release candidate; complete but not approved as operational.
- `rev` indicates very minor revision that does not impact schema validation.

The regular expression notation used above is from **[W3-XML]** `#sec-notation`.

Note that the absence of a `status` string in the version number indicates that the version has been baselined and published.

The regular expression in **[Rule 4-5]**, above, allows the following example version numbers:

- `1`
- `1.2`
- `1.3.1.0`
- `1.2alpha13`
- `199.88.15rev6`

There are two implications in **[Rule 4-5]**, above. The first is that in some cases this version scheme

implies and confirms a chronology of releases. For example, a given product labeled version 2.3 must have been released before the same product labeled 2.3.1. Therefore, version 2.3.1 is more current than version 2.3.

However, this is a multi-series version scheme, and chronological relationships exist only within a given series. So, for example, nothing can be said about a chronological relationship between versions 2.2.4 and 2.3. This is because version 2.2.4 is in a different series (i.e., 2.2) and could actually have been released after 2.3. Figure 4-1, *Example versioning system*, below, illustrates a system of versions that uses the numbering scheme of **[Rule 4-5]**, above.

Figure 4-1: Example versioning system

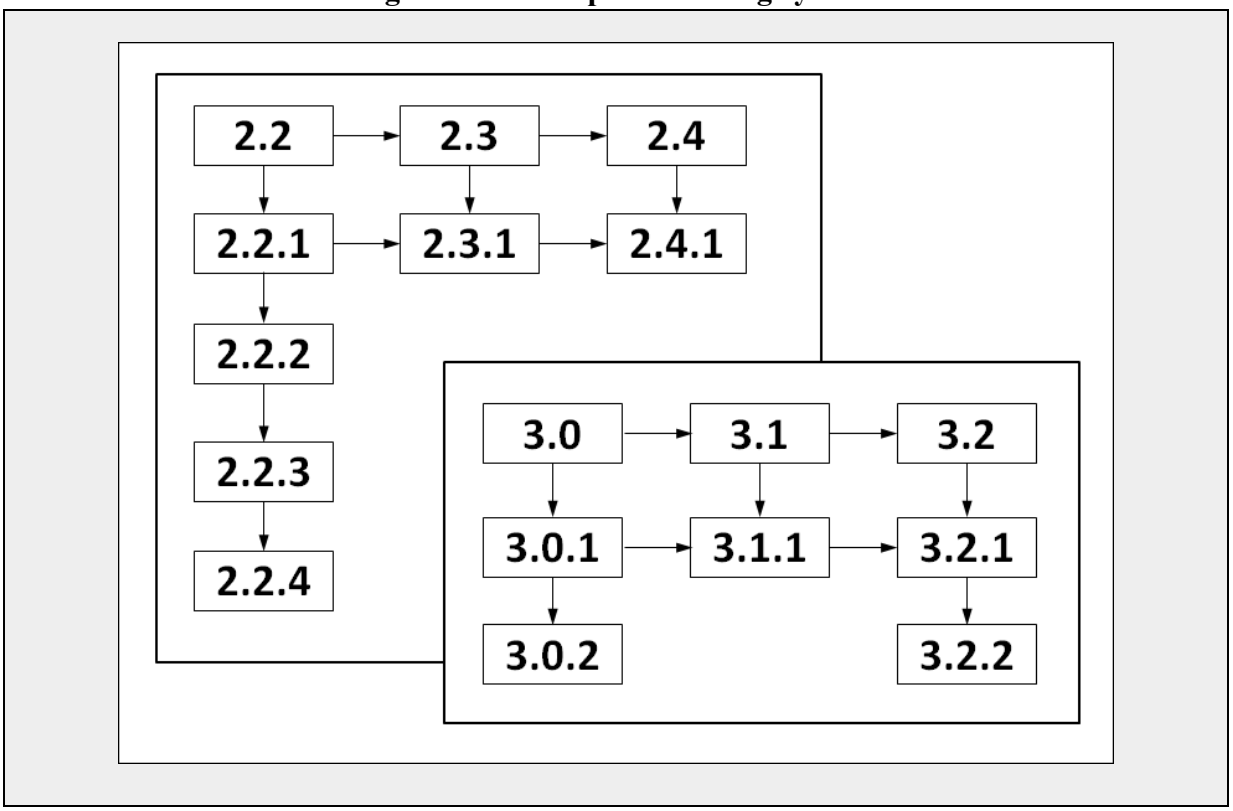


Figure 4-1, *Example versioning system*, above, illustrates eight different version series. Within this illustration these are the only sequences that have chronological relationships that can be identified through version numbers.

- Series 2 is {2.2, 2.3, 2.4}
- Series 3 is {3.0, 3.1, 3.2}
- Series 2.2 is {2.2(.0), 2.2.1, 2.2.2, 2.2.3, 2.2.4}
- Series 2.3 is {2.3(.0), 2.3.1}
- Series 2.4 is {2.4(.0), 2.4.1}
- Series 3.0 is {3.0(.0), 3.0.1, 3.0.2}
- Series 3.1 is {3.1(.0), 3.1.1}
- Series 3.2 is {3.2(.0), 3.2.1, 3.2.2}

The second implication of **[Rule 4-5]**, above, is that pre-releases are easily identified by the strings `alpha`, `beta`, and `rc`. These strings are simple visible indicators of MPD status or stage of development.

This specification places no further restrictions or meaning (implied or otherwise) on a version number.

Authors have the option to use integers between dots to indicate degree of compatibility or other relationships between versions as needed. For example, for a given MPD, the author may declare that if an instance validates to version 4.2.3, then it will also validate to version 4.2. Such a claim is acceptable. However, this specification does not imply any such relationships. Any meaning assigned to version sequence by an authoritative source should be unambiguously documented within the MPD.

[Rule 4-6]

MPD version numbers within a version series do NOT imply compatibility between versions. Compatibility between or among MPD versions MUST be explicitly stated in documentation.

Note that an author who updates an existing MPD to a new version may choose the version number based on its previous version number or not, as long as it follows the version number syntax.

Version number syntax applies to MPDs only; there is no mandate to apply this syntax to artifact versioning. To do so is optional.

4.2.2. URI Scheme for MPDs

To facilitate MPD sharing and reuse the assignment of a URI (Uniform Resource Identifier) to each MPD is essential.

[Rule 4-7]

Every MPD MUST be assigned a valid `http` URI.

This specification follows **[RFC3986 URI]** which defines the syntax and format for a URI. However, this specification also restricts an MPD URI to a URL and does not allow a URN (Uniform Resource Name) to be assigned to an MPD.

Here is a typical example of an http URI: `http://www.abc.org/niem-iepd/order/2.1.2rev3/`

Note that **[Rule 4-7]**, above, explicitly states that a URI assigned to an MPD must be valid. This means that the person or organization assigning the URI either is the registrant of the domain name, or has authority from the registrant to assign this URL as an MPD URI. In the example above, `www.abc.org` is the domain name (between the second and third "/"). There is no requirement for a URL assigned to an MPD to [resolve] to any particular Internet resource or to [resolve] at all. However, it is always good practice for such a URL to [resolve] to the resource it represents, the directory it resides in, or to documentation for that resource. See `http://www.w3.org/Provider/Style/URI.html`

The MPD version number (i.e., the value of the `mpdVersionID` in Appendix A, *MPD Catalog XML Schema Document*, below) is essential to its unique identification. Incorporation of the version number within the MPD URI provides a simple visual (as well as machine readable) means of identifying one of the most fundamental relationships between MPDs, i.e., that one is a different version of another. Another advantage to this technique is that different versions of an MPD will generally group together

in a standard sorted ordering.

[Rule 4-8]

The URI for an MPD MUST end in its version number concatenated with a forward slash character ("/").

And finally, note that Appendix A, *MPD Catalog XML Schema Document*, below, defines a mandatory attribute for the `mpdURI`, `mpdName`, `mpdClassCode`, and `mpdVersionID`. Since the ending substring of an MPD URI must be its version ID (followed by forward slash), then the mpd-catalog duplicates the MPD version ID in two locations. By design, Section 7.1, *MPD File Name Syntax*, below, intentionally duplicates these attribute values in the MPD file name. There are two reasons for this design. First, software tools are expected to build and process mpd-catalogs. Instead of forcing tool developers to parse the URI just to retrieve name, version, and class, the mpd-catalog separates these data items as XML attributes. Second, duplication of that key metadata in both the URI and the file name facilitates fast visual recognition of an MPD, rather than requiring that a user open the archive, open its `mpd-catalog.xml`, and scan XML content to locate the data.

4.2.3. URI Scheme for MPD Artifacts

Given the URI for an MPD, a URI also exists for each artifact in that MPD. Again, this specification follows **[RFC3986 URI]** and employs a fragment identifier to designate an artifact URI. Each file artifact or set (directory) artifact is uniquely identified by its path name relative to the [MPD root directory]. The URI for an MPD artifact is the concatenatation of (1) the MPD URI, (2) the "#" character, and (3) the path name of the artifact.

[Rule 4-9]

The URI reference to an individual MPD artifact from another resource is the concatenation of

- The URI of the MPD that contains the artifact.
- The crosshatch or pound character ("#").
- A fragment identifier that is the locally unique path name (string) of the artifact relative to the [MPD root directory].

Thus, each MPD artifact (file or set) has a globally unique URI that can be referenced from other external resources as needed. Example artifact URIs include:

- `http://example.gov/niem-iepd/pmix/3.0/#subset/niem-core.xsd`
- `http://example.gov/niem-iepd/pmix/3.0beta2/#extension/ext-1.1.xsd`
- `http://example.gov/niem-iepd/pmix/3.0/#documentation/user-manual.docx`
- `http://example.gov/niem-iepd/pmix/3.0/#application-info/` (a set artifact)
- `http://example.gov/niem-iepd/pmix/3.0/#iep-sample/query/` (a set artifact)
- `http://www.abc.org/niem-iepd/order/2.1.2rev3/#extension/request4.xsd`

Here is one simple scenario for use of an artifact URI within the mpd-catalog. Consider two different IEPDs with the following URIs:

1. `http://example.gov/niem-iepd/pmix/3.0/`
2. `http://www.abc.org/niem-iepd/order/2.1.2rev3/`

The author of IEPD (1) has decided to reuse (as-is) the `extension/request1.xsd` artifact in IEPD (2). He/she can optionally create an mpd-catalog `ExtensionSchemaDocument` entry for this artifact (assuming it is an extension schema document), and add the attribute:

```
@externalURI="http://www.abc.org/niem-
iepd/order/2.1.2rev3/#extension/request4.xsd"
```

Additional `@externalURI` attributes can be optionally added to this entry if the author knows of other uses of this same artifact in other MPDs and wishes to acknowledge them.

Note that a URI does not have the same meaning as namespace. Do not rely on namespaces for artifact URIs. Recall that the namespaces used in a schema document subset derived from a NIEM release are identical to the namespaces of the release itself. Furthermore, an IEPD or an EIEM may contain multiple subsets. The non-uniqueness of NIEM namespaces implies that they cannot be used as URIs for MPD artifacts.

> **[Rule 4-10]**
>
> NIEM namespaces MUST NOT be used as URIs for MPD artifacts.

Later in Section 4.5, *XML Catalogs*, below, we will describe the use of **[XML Catalogs]** to correlate and [resolve] namespaces to their corresponding local URIs.

4.2.4. MPD Artifact Lineage

An important MPD business requirement is transparency of lineage. MPDs internally facilitate identification of the relationships that may exist among their artifacts, families, versions, adaptations, specializations, generalizations, etc. The URI scheme for MPDs and artifacts as well as the mpd-catalog make this possible.

The mpd-catalog provides a `Relationship` element with three attributes (`resourceURI`, `relationshipCode`, and `descriptionText`) to identify the pedigree of an MPD. There are many ways that one MPD may relate to another. This makes it extremely difficult to specify a fixed set of values that can objectively define an exact relationship between a pair of MPDs. Therefore, the optional `descriptionText` attribute is provided to further explain the nature of any of the eight `relationshipCode` values available {`version_of`, `specializes`, `generalizes`, `deprecates`, `supersedes`, `adapts`, `conforms_to`, `updates`}. In some cases, the value of `relationshipCode` may be generic enough to require a more detailed explanation in `descriptionText` (for example, if its value is `adapts`).

The mpd-catalog also enables an author to record a fine-grained pedigree between MPDs when reusing artifacts from other MPDs. By default each artifact identified in an mpd-catalog has a globally

unique URI (using a fragment reference) that can refer to it. An MPD author signifies reuse of a given artifact by entering the URI for that artifact in the optional `externalURI` attribute within the appropriate `FileType` or `FileSetType` elements.

Some MPDs are designed for more extensive reuse than others. For example, families of IEPDs are expected to reuse a given EIEM. In such cases, the mpd-catalogs for these IEPDs and the corresponding EIEM may overlap in or duplicate a large number of metadata and references. This is expected. The mpd-catalog can contain many references to and semantics for artifacts and MPDs. Correct and consistent use of these references and semantics will create networks of related MPDs so that tools can locate, parse, and process them as needed and when available in shared repositories.

4.3. Change Log

4.3.1. Change Log for Releases and Core/Domain Updates

Although the version identifier is useful for a fast visual indication of the state of an MPD, it only provides a general indication that the MPD has changed. There is no indication of the volume, complexity, or impact of changes applied since a previous version. A *change log* provides a more specific accounting of changes from one version to another.

[Definition: change log]

A formal (for releases, core updates, domain updates) or informal (for IEPDs and EIEMs) artifact that accounts for changes applied to an MPD since its previous version (or versions).

Once published, NIEM releases always exist. This ensures that IEPDs and EIEMs built from a given release will always be usable, and may be updated to a new NIEM release only when convenient or absolutely necessary to take advantage of new or modified data components. Though not encouraged, nothing prohibits a developer from building an IEPD based on a NIEM release that is older than the most current version. There may be potential disadvantages related to interoperability levels achievable with others developing to the latest release. Nonetheless, an older version might meet the business needs of a particular organization quite well.

In spite of this built-in stability, the NIEM architecture is designed to evolve as requirements change. New versions of reference schema document sets such as NIEM releases, core updates, and domain updates can have significant impacts on future IEPDs and EIEMs. Developers must understand in detail how changes will affect their IEPD and EIEM products and the tools used to build them. To work effectively, tools for domain content development, impact analysis, migration between releases, etc. must be able to digest formal change logs. A formal change log is also essential to efficiently process and integrate new and changed content into NIEM for new releases, and to simultaneously maintain multiple versions of NIEM for users. All of the foregoing reasons dictate that NIEM require a normative change log for reference schema document sets.

[Rule 4-11]

Every MPD that is a reference schema document set (i.e., NIEM releases, core updates, and domain updates) MUST contain an XML change log document that:

- Validates with the NIEM change log schemas `mpd-changelog.xsd` and `niem-model.xsd`. (Note these are base filenames; actual filenames also contain a version number; for example, `mpd-changelog-1.0.xsd`.)
- Records changes to previous reference schema documents that this MPD represents.
- Bears the file name `changelog.xml`.
- Resides in the [MPD root directory].

The current version of `mpd-changelog.xsd` is available here:

```
http://reference.niem.gov/niem/resource/mpd/changelog/
```

The current version of `niem-model.xsd` which describes the NIEM conceptual model is available here:

```
http://reference.niem.gov/niem/resource/model/
```

Since the schemas are the authority for a release or update and because almost all tool support depends on the schemas, the change log is only designed to audit transactional changes to the reference schema documents. There is no provision for logging changes to support documentation or other non-schema artifacts. Non-schema changes are generally handled non-normatively in the form of release notes.

4.3.2. Change Log for IEPDs and EIEMs

IEPD and EIEM change log requirements are less strict and are not required to conform to the naming and XML schema specifications in **[Rule 4-11]**, above. However, a change log is still required.

[Rule 4-12]

Every MPD that is an IEPD or EIEM MUST contain a change log artifact that:

- Records changes to previous IEPD or EIEM schemas that this MPD represents.
- Has a file name that begins with the substring "changelog".
- Resides in the [MPD root directory].

This rule does not specify the format for an IEPD or EIEM change log. This is left to the discretion of the author. While use of `mpd-changelog.xsd` is encouraged for IEPD and EIEM schemas, it is not required. Relaxing the change log format encourages and facilitates easier and more rapid development. IEPDs and EIEMs are developed by a variety of NIEM domains, organizations, and users; and they are intended to specify implementable exchanges. As a result, IEPDs and EIEMs may contain both documentation artifacts and machine readable application artifacts in a large variety of formats. A consistent standard change log would be very difficult to specify.

The initial version of an IEPD or EIEM would not normally require a change log. However, for

consistency of validation and to help facilitate automatic processing of IEPDs and EIEMs by tools:

[Rule 4-13]

The initial version of an IEPD or EIEM MUST contain a change log artifact with at least one entry for its creation date.

Finally, if the `mpd-changelog.xsd` specification is used for IEPD/EIEM schema changes, then it is potentially possible that such an MPD will need a second change log if the author wants to accommodate documentation or other changes not related to schemas (since `mpd-changelog.xsd` cannot be extended to accommodate such changes). If this is the case, then the following rule applies:

[Rule 4-14]

If an IEPD or EIEM contains more than one change log artifact, then each change log artifact MUST:

- Have a file name that begins with the substring `changelog`.
- Reside in the [MPD root directory].

4.4. Master Document

This section is only applicable to IEPDs and EIEMs.

[Definition: master document]

An informal documentation artifact contained in an IEPD or EIEM that serves as the initial general source of human readable descriptive or instructional information. A *master document* may index or reference other more specific documentation or other explanatory materials within the MPD.

The *master document* is similar to a `readme` file. It is only required for IEPDs and EIEMs since these MPDs are allowed the greatest design flexibility, can be developed and implemented different ways, and are not centrally managed. On the other hand, releases and domain updates have fairly restrictive rules to obey, standard documentation for how to use them, and are centrally managed.

[Rule 4-15]

An IEPD or an EIEM MUST contain a master document located in the [MPD root directory] whose filename begins with the substring `master-document`.

The master document may replicate some of the metadata in the mpd-catalog. However, the mpd-catalog is intentionally designed to be efficient, easily to parse, and minimal. It is intended for search, discovery, registration, and Web page generation, and not to support various types of detailed technical prose often required for human understanding.

The primary purposes of the master document include:

- To help facilitate understanding and reuse of IEPDs and EIEMs.
- To ensure that fundamental and detailed business-level information about an IEPD or EIEM are documented for human understanding.
- To ensure the IEPD or EIEM author has considered and conveys such fundamental information.
- To provide an initial source within an IEPD or EIEM for human consumable documentation (similar to a `readme` file) and/or references to other business or technical documentation needed for understanding.

The master document is not intended to be the only source of written documentation for an MPD (though it can be). It is expected to be the initial resource that references and coordinates all others whether physically present in the MPD or linked by reference. Many organizations have their own customized formats and operating procedures for documenting their work and products. This specification does not attempt to standardize master document format or layout. Only the file name and relative path within the MPD archive are strictly specified. The following section will generally describe minimal content that should be in the master document. This guidance is non-normative, so adherence is a subjective judgment by the author.

4.4.1. Master Document Content

This section is neither a cookbook nor a normative specification for a master document. It simply suggests typical topics that a master document should or might address, and provides some non-normative guidance.

The master document should help another user or developer to understand the content and use of an IEPD or EIEM, as well as determine potential for reuse or adaptation. It should describe what implementers need to understand and what the author considers is important to understanding an IEPD or EIEM. There is no limit or constraint on its content.

At a minimum, the master document should contain several fundamental elements of information about the MPD:

- Purpose of this MPD.
- Scope of its deployment, usage, and information content.
- Business value and rationale for developing it.
- Type of information it is intended to exchange (in business terms).
- Identification of senders and receivers (or the types of senders and receivers).
- Typical interactions between senders, receivers, and systems.
- References to other documentation within the MPD, and links to external documents that may be needed to understand and implement it.

Many document formats (e.g., Microsoft Word) can display hot links to local files within the MPD archive as well as URLs to files on the Internet. Employing such a format is highly recommended but not mandatory.

[Rule 4-16]

A NIEM IEPD or EIEM master document SHOULD (at a minimum) describe the MPD purpose, scope, business value, exchange information, typical senders/receivers, interactions, and references to other documentation.

MPD documentation types and formats will vary with the methodologies and tools used to develop them. Most of this documentation will likely be typical of that generated for data-oriented software projects. Some documentation may only require sections in the master document. Other documentation may be more suitable as separate artifacts that are referenced and explained by a section in the master document (such as diagrams, large tables, data dictionaries, test results/reports, etc.). The following are some common examples of sections in or separate artifacts associated with the master document:

- Executive summary (especially for lengthy master documents>
- Use cases
- Business processes
- Business requirements
- Business rules
- Metadata security considerations
- Domain model design specifications and documentation and/or diagrams
- Data dictionary
- Testing and conformance
- Development tools and methodologies used
- Implementation guidance (particularly important for a complex IEPD with multiple subsets or IEP root elements)
- Security considerations
- Privacy considerations (e.g., Personal Identifiable Information)
- Types of implementations

- If an IEPD employs multiple subsets:

    - When, where, and how these are used
    - How these are coordinated in the implementation
    - Caveats regarding duplicate data components (which can occur with multiple subsets)

- If an IEPD employs multiple IEP conformance targets:

    - Purpose of each and when it should be used
    - How these are coordinated during the runtime preparation and transmission of IEPs

4.5. XML Catalogs

This section is applicable to all MPDs. However, it is of particular importance to IEPDs and IEP validation (to be covered in more detail in Section 4.6, *Information Exchange Packages*, below.

[XML Catalogs] are XML documents that describe a mapping between external entity references and locally-cached equivalents. They are used to [resolve] XML schema document target namespaces

to local URIs. This is especially useful when assembling an XML schema from an XML schema document set. Some validators (e.g., `Xerces`) and other tools utilize *XML catalogs* for this purpose.

> **[Definition: XML catalog document]**
>
> An XML document defined by the semantics of **[XML Catalogs]**.

The **[NIEM SSGT]** (for NIEM 3.0) automatically adds an `xml-catalog.xml` artifact to each schema document subset it generates. The NIEM 3.0 release also includes such an artifact. These [XML catalog documents] are provided for user convenience in the case these schema document sets must be assembled into a schema.

IEPD authors must employ [XML catalog documents] within IEPDs to facilitate validation of IEPs.

Assembling a schema or building an [XML catalog document] from the XML schema documents of non-conformant external standards that contain `xs:include` statements can be problematic. Be aware that if an xml-catalog (resulting from processing a set of external XML schema documents) contains any two `uri` element entries with identical namespaces, then that xml-catalog cannot be used for XML validation. It will have to be modified to ensure that each namespace resolves to one and only one unique [XML catalog document] `uri` attribute value.

In order to support schema assembly for the purpose of XML validation, the following rule requires that the namespaces of all XML schema documents used within an IEPD [resolve] to a locally-unique artifact:

> **[Rule 4-17]**
>
> An IEPD MUST [resolve] each namespace it uses to a locally unique URI through one or more [XML catalog documents].

This rule implies that `NextCatalog` elements may be used within [XML catalog documents] to connect them and control their parsing sequence. An IEPD must contain at least one [XML catalog document] because it is the only MPD that can specify an IEP **[Definition: Information Exchange Package (IEP)]** and provide validation instructions that would require schema assembly from XML schema documents. Section 4.6, *Information Exchange Packages*, below, provides more specifics about using [XML catalog documents] within IEPDs.

4.6. Information Exchange Packages

This section only applies to IEPDs. An IEPD is the only MPD that defines IEPs **[Definition: Information Exchange Package (IEP)]**. An IEPD does this by declaring (either implicitly or explicitly) one or more *IEP conformance targets*.

> **[Definition: IEP conformance target]**

A class or category of IEPs which has a set of one or more validity constraints and a unique identifier. Every IEP is an instance of one or more IEP conformance targets.

This definition requires that a IEP conformance target be associated with a unique identifier, a *Conformance Target URI* that distinguishes it from all other IEP conformance targets. Similar to a URI for an MPD artifact, construct a *conformance target URI* by concatenating the IEPD's http URI, the pound character (`#`), and a locally unique (within the IEPD) `NCName` **[W3C XML Schema Structures]**.

[Definition: IEP conformance target URI]

A globally unique identifier for an IEP conformance target declared in an IEPD, formed by concatenating:

1. the IEPD URI
2. the pound character (`#`) and
3. a locally unique `NCName` per **[W3C XML Schema Structures]**

The foregoing definition requires that an IEP conformance target class have a URI. As a result, the following rule is also required:

[Rule 4-18]

An `IEPConformanceTarget` MUST be assigned a locally unique NCName value for its `structures:id` attribute.

An IEPD defines IEP conformance targets by explicitly and formally declaring them within its mpd-catalog. The rule above ensures that conformance targets can be referenced between IEPDs and not only within an IEPD.

The following subsections the concepts, artifacts, and procedures for declaring and identifying IEP conformance targets in IEPDs.

4.6.1. Schema Validation

NIEM employs the W3C XML Schema Definition (XSD) Language (**[W3C XML Schema Structures]** and **[W3C XML Schema Datatypes]**), one of several XML schema definition languages designed to define an instance XML document and enable its validation. In general, an instance XML document is valid against a particular XML schema if it obeys or conforms to the constraints imposed by that schema (**[W3C XML Schema Structures]** `2.5 Schema-validity and documents`).

So, a NIEM IEPD is an MPD that contains a set of XML schema documents, that are assembled into an XML schema (after processing xml-catalogs to [resolve] `xs:import` statements and similar XML

Schema constructs). In turn, the resulting XML schema can be used to validate one or more instance XML documents (i.e., NIEM IEPs **[Definition: Information Exchange Package (IEP)]**) for NIEM conformance.

NIEM is based on XML Schema, and so the term "schema validation" usually refers to "XML Schema validation". However, an IEPD author may also choose to include artifacts to validate with other types of schemas or rules, including but not limited to **[ISO Schematron]** and **[ISO RelaxNG]**. IEPD authors may also include artifacts for NIEM constraint schema validation, which, of course, is XML Schema validation (See Section 3.5, *Constraint Schema Documents and Document Sets*, above.

4.6.2. Declaring Validity Constraints

Explicit declaration of validity constraints is far more flexible than relying on convention. Declaring validity constraints in the mpd-catalog, frees an IEPD author from having to follow conventional IEPD organization. Many standard validity constraints can be controlled from `IEPConformanceTarget` elements within the mpd-catalog without the need to standardize IEPD directory structure or hunt for validation artifacts. Instead, the `IEPConformanceTarget` element identifies the conformance target, the type of validation, and the location of the necessary validation artifact(s). It can also identify IEP samples known to satisfy the validity constraints.

> **[Rule 4-19]**
>
> An IEPD MUST explicitly declare all intended validity constraints by employing the `IEPConformanceTarget` element in its `mpd-catalog.xml` artifact.

Appendix A, *MPD Catalog XML Schema Document*, below, provides XML elements for various validity constraints. Each element is in the `substitutionGroup` for the abstract element `ValidityConstraint`. Note that there may exist multiple ways to declare the same validity constraint with these elements. This rule only requires that validity constraints be declared once in a single form. For example, it may be possible to use the `HasDocumentElement` and the `ValidToXPath` elements to declare the same XML document elements. However, it is only required that an IEPD author use one or the other.

Recall that this section is not applicable to an MPD that is an EIEM, because it does not define an information exchange. That said, there may be good rationales for an EIEM author to provide validity constraints with an EIEM. However, this is not a required for an EIEM.

The following subsections explain in more detail the purpose and use of each `IEPConformanceTarget` validity constraint.

4.6.2.1. ValidToXPath

`ValidToXPath` is the most generic of the constraints provided. Its purpose is to ensure that some given condition is satisfied within an IEP. The condition is defined by an XPath expression contained in the `xPathText` attribute. If the XPath expression applied to a target instance XML document returns a Boolean value of TRUE, then the condition is satisfied by that XML document.

This validity constraint is useful for a variety of purposes. For example, an IEPD author may require that a given `IEPConformanceTarget`) must contain a particular element with a particular attribute whose value is an integer greater than some required minimum. An XPath expression can validate this.

Of course, `ValidToXPath` can also employ a simple XPath expression to validate that an IEP is rooted with an intended XML document element. However, there are other validity constraints that can do this as well. The IEPD author may choose the configuration.

4.6.2.2. XMLSchemaValid

Because NIEM is based on XML Schema, then `XMLSchemaValid` will likely be employed by almost all IEPDs. This constraint simply ensures that an IEP artifact is schema valid to an XML schema described by and assembled with an XML Catalog. The starting XML Catalog artifact is identified by the `XMLCatalog` element.

4.6.2.3. SchematronValid

`SchematronValid` is similar to `XMLSchemaValid`, but uses a `SchematronSchema` element to identify the Schematron rule file that applies to the IEP.

4.6.2.4. RelaxNGValid

`RelaxNGValid` is similar to the previous two validity constraints, but uses a `RelaxNGSchema` element to identify the RelaxNG schema file to which the IEP must validate.

4.6.2.5. HasDocumentElement

`HasDocumentElement` is a validity constraint that identifies all intended XML document elements for an IEP conformance target. This constraint ensures that an IEP artifact is rooted by one XML document element that is a member of the list of elements in its `qualifiedNameList` attribute. This is a common validity constraint employed by simple IEPDs that declare one or more intended XML document elements.

Note that this validity constraint only declares XML document elements. If an IEPD defines IEPs for payloads and envelopes (i.e., one IEP encapsulates another), then other validity constraints will be required (such as `ValidToXPath`) to validate conformance targets that are not necessarily XML document elements.

4.6.2.6. ConformsToConformanceTarget

`ConformsToConformanceTarget` enables an IEPD author to effectively subclass and relate conformance target classes. For example, using this constraint, a given conformance target class defined by an `IEPConformanceTarget structures:id="A2"` can be required to also conform to another class `structures:id="A1"`. This creates an *IS-A* relationship. We say that `A2` *is an* `A1`, or that `A2` is a specialization of `A1`.

Conformance target classes are related through the `ConformsToConformanceTarget/conformanceTargetURI` attribute. Recall in **[Definition: IEP**

conformance target URI] that this URI is formed by the concatenation of the URI of the IEPD (`mpdURI`) itself, the pound character ("`#`"), and the value of the conformance class `structures:id`.

4.6.2.7. ConformsToRule

Sometimes it is not possible to formally declare an executable validity constraint. For example, we can mandate that a data component definition must be present, must be in English, and must follow **[ISO 11179-4]**. Validating that text is present is easy, and validating that is in English is more difficult, but validating that it obeys **[ISO 11179-4]** is intractable. Thus, `ConformsToRule` provides an IEPD author with English text representation as an alternative when it is not possible or not easy to define more formal validation rules or validity constraints.

4.6.3. IEP Sample XML Instance Documents

Sample IEP XML instance documents are representations of actual or example exchange data instances and can be extremely valuable artifacts in an IEPD. Sample IEPs can:

- Help an IEPD implementer to understand the original intent of the IEPD author.
- Be used by an implementer as a data point for validation of IEP conformance targets.
- Indicate or imply IEPD quality.

For these reasons, IEP samples are required for IEPDs:

> **[Rule 4-20]**
>
> An IEPD MUST contain at least one IEP sample XML document instance that exemplifies each declared `IEPConformanceTarget`. If applicable and appropriate, one IEP sample MAY exemplify multiple conformance targets.

Note that this rule requires that each IEP conformance target be covered by at least one IEP sample document instance. This does not necessarily mandate a different IEP sample for each IEP conformance target. It may be possible, and is therefore acceptable, for a given IEP sample to serve as an example of one or more IEP conformance targets.

The purpose of this rule is not to provide a test for all IEP permutations that might be possible given the schema definitions and validity constraint declarations; rather, it is to encourage IEPD authors to test their own designs, and to provide implementers with examples for additional understanding, guidance, and testing. To the extent possible, IEPD authors should strive to include sample IEPs that (1) capture real world business cases of data exchanges, and (2) exercise as many data components and validity constraints as possible. Where it makes sense, an IEPD author should strive to provide enough sample IEPs to exercise all the XML document elements (or payload root elements). If a single IEP cannot provide enough example coverage, an author may include multiple IEPs (but is not required to do so).

Each sample IEP usually illustrates a single view of the data based on a chosen set of conditions. Other views based on different conditions likely exist. An implementer will likely still need to review the IEPD documentation to ensure understanding of all potential conditions. Therefore, as appropriate, the

author should not rely exclusively on sample IEPs to convey implementation understanding, since they will not likely account for all possible permutations.

This specification also encourages (but does not mandate) inclusion of sample IEPs in other MPDs such as EIEMs and domain updates; these can sometimes provide valuable insight into the intent and usage of new, extended, or changed data components.

5. MPD Resolution, Existence, and Validation Rules

NOTES for rules:

- (X) to the right of the rule number indicates the applicable conformance target "X".
- Schema document sets can be defined by directory, list, or xml-catalog.

All MPDs use URIs to identify, find, or acquire artifacts and other resources. As such, the following definition for resolving URIs will be useful to validation rules.

[Definition: resolve URI]

An function (or action) that takes a URI string of the form `xs:anyURI` and returns the resource it identifies. If the resource does not exist, then this function fails. If a resource is remote (e.g., a URL on the Internet or a URN of an unknown location), then this function (or action) may require human intervention to complete.

[NIEM NDR], `§15.3.6.1 Reference Elements` defines a [reference element] as follows:

[Definition: reference element]

An XML element that refers to its value by a reference attribute instead of carrying it as content

The [MPD catalog document] reuses NIEM and so it conforms to NIEM. This means that an author may use one or more [reference elements] from various locations to refer to a single content bearing instance of the same element (with a unique `structures:id`). The definition of [resolve] and the conformance target rules that follow assume content bearing elements. Therefore, if a rule applies to a conformance target with a URI attribute whose owning element is in [reference element] form, then URI resolution will be applied at the site of the content-bearing element form referenced.

[Rule 5-1] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute MUST [resolve] to an MPD artifact.

[Rule 5-2] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:XMLCatalog` element MUST [resolve] to an [XML catalog document].

[Rule 5-3] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:MPDChangeLog` element MUST [resolve] to a [change log].

[Rule 5-4] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:MasterDocument` element MUST [resolve] to a [master document].

[Rule 5-5] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:IEPSampleXMLDocument` element MUST [resolve] to an [XML document].

[Rule 5-6] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:BusinessRulesArtifact` element MUST [resolve] to a [business rule schema] or [business rules] artifact.

[Rule 5-7] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:ExternalSchemaDocument` element MUST [resolve] to an [XML schema document].

[Rule 5-8] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:ReferenceSchemaDocument` element MUST [resolve] to NIEM [reference schema document].

[Rule 5-9] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:ExtensionSchemaDocument` element MUST [resolve] to a NIEM [extension schema document].

[Rule 5-10] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:SubsetSchemaDocument` element MUST [resolve] to a NIEM [subset schema document].

[Rule 5-11] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:Wantlist` element MUST [resolve] to a [NIEM wantlist] XML document.

[Rule 5-12] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:SchematronSchema` element MUST [resolve] to a [Schematron schema].

[Rule 5-13] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:RelaxNGSchema` element MUST [resolve] to a [RelaxNG schema].

[Rule 5-14] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:SchemaDocumentSet` element MUST [resolve] to an [XML schema document] set.

[Rule 5-15] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:ConstraintSchemaDocumentSet` element MUST [resolve] to a NIEM [constraint schema document set].

[Rule 5-16] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:ReferenceSchemaDocumentSet` element MUST [resolve] to a NIEM [reference schema document set].

[Rule 5-17] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:SchematronSchema` element MUST [resolve] to an [Schematron schema] .

[Rule 5-18] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:RelaxNGSchema` element MUST [resolve] to an [RelaxNG schema] .

[Rule 5-19] (MPD)

Within the [MPD catalog document] the value of a `c:pathURI` attribute owned by a `c:RequiredFile` element MUST [resolve] to a file artifact.

[Rule 5-20] (IEP)

Within the [MPD catalog document] the value of a `c:XPathText` attribute owned by a `c:ValidToXPath` element applied to a candidate IEP (an [XML document]) will evaluate to an effective Boolean value (EBV). The candidate is a valid IEP *only if* that EBV equals `"true"`.
[W3C XPath 2.0] `http://www.w3.org/TR/2010/REC-xpath20-20101214/#id-ebv`

[Rule 5-21] (IEP)

Given an [MPD catalog document] with a `c:qualifiedNameList` attribute owned by a `c:HasDocumentElement` element, a candidate IEP is a valid IEP, *only if* its XML document

element is a member of that list of *QNames*.

[Rule 5-22] (IEP)

Given an [MPD catalog document] with a `c:qualifiedNameList` attribute owned by a `c:HasElement` element, a candidate IEP is a valid IEP, *only if* it contains an XML element that is a member of that list of *QNames*.

[Rule 5-23] (IEP)

Given a `c:IEPSampleXMLDocument` whose parent is `c:IEPConformanceTarget`, the artifact resolved by `c:pathURI` MUST be valid to the `c:IEPConformanceTarget` validity constraints.

[Rule 5-24] (XML-Catalog)

Within an [XML catalog document] the value of the `er:uri` attribute owned by the `er:URI` element MUST [resolve] to an [XML schema document].

[Rule 5-25] (XML-Catalog)

Within an [XML catalog document], given the [XML schema document] resolved by the value of the `er:uri` attribute owned by the `er:URI` element, the [XML schema document] target namespace MUST equal the value of the `er:name` (a namespace string) attribute owned by the `er:URI` element.

[Rule 5-26] (Full-NIEM)

Within an [MPD catalog document] the value of a `c:externalURI` attribute owned by an element that is a `c:FileType` (or a type derived from it) MUST [resolve] to a resource.

[Rule 5-27] (Full-NIEM)

Within an [MPD catalog document] the value of a `c:resourceURI` attribute owned by a `c:Relationship` element MUST [resolve] to a resource.

Potential rule classifications defined:

[Definition: well-formed MPD]

Satisfies the following criteria:

- Is a valid **[PKZIP]** archive file.
- Contains one and only one `mpd-catalog.xml` artifact.
- Its `mpd-catalog.xml` artifact resides in the [MPD root directory].
- Its `mpd-catalog.xml` artifact is XML Schema valid to Appendix A, *MPD Catalog XML Schema Document*, below.
- Adheres to the validity rules defined in Section 5, *MPD Resolution, Existence, and Validation Rules*, above, (this section).

[Definition: complete MPD]

Satisfies all rules and mandatory requirements for:

- this NIEM MPD Specification (**[NIEM MPD Specification]**)
- **[NIEM NDR]**
- **[NIEM Conformance]**

[Definition: full NIEM MPD]

Satisfies the following criteria:

-

6. Optional MPD Artifacts

Aside from the required artifacts, MPD content is relatively flexible. A variety of other optional documentation files may be incorporated into an MPD. When applicable, these may include (but are not limited to) files that describe or explain:

- Implementation details (hardware, software, configuration, etc.)
- Use of multiple root elements
- Use of multiple subsets or mixed releases
- How to use/reuse an MPD for various purposes (such as Web Services)
- Rationales and/or business purposes

In addition to documentation artifacts, a variety of other optional files can be added to an MPD to facilitate tool support and make reuse, adaptation, and/or implementation easier. These are often files that are inputs to or outputs from software tools. Examples include content diagrams, content models

in tool-specific formats, and business rules (either formal or informal representations).

Another optional artifact that is encouraged, especially for IEPDs, is a conformance report or other evidence of quality. In the future, as NIEM processes and tools mature, conformance and quality reports and a corresponding certificate may become required artifacts. For now, inclusion of a conformance report is at the discretion of the author or sponsor. Though clearly, such reports can only increase confidence in MPDs that contain them.

An MPD author may include any files believed to be useful to understand, implement, reuse, and/or adapt an MPD.

An MPD of relatively simple content and scope may only need to contain the minimum mandatory artifacts required by this specification in order to understand and implement it. (See Appendix C, *MPD Artifacts*, below, for a listing of the mandatory and common optional artifacts for each type of MPD.)

Files vary widely in format and are often specific to the tools an author uses to parse, consume, or output them. Therefore, if tool-specific files are included in an MPD, it is also a good practice to include copies of those files in formats that display with standard Web browsers or other cost-free, publicly available viewing tools (e.g., ASCII text, PDF, CSV, HTML, JPG, GIF, PNG). This guidance is intended to encourage and facilitate maximal sharing and distribution of MPDs; it does not prohibit and is not intended to discourage the inclusion of other file formats.

In particular, this specification does not discourage use of Microsoft file formats for documentation and other optional artifacts. Microsoft Office products are in common use, and free viewers are available for many of them (See `http://office.microsoft.com/en-us/downloads/office-online-file-converters-and-viewers-HA001044981.aspx`).

6.1. NIEM Wantlist

A NIEM schema document subset is often associated with a NIEM *wantlist*. A *wantlist* is an abbreviated XML representation of a NIEM schema document subset, and identifies only the data components a user selected (as requirements) to build a schema document subset. To reconstruct the complete schema document subset there are usually a number of additional data components that the user selections depend upon. These must be computed from the appropriate NIEM reference model and added to reconstruct the complete schema document subset. For example, a user may select `nc:Person` for the subset. In this case, the wantlist will only contain that component, but the associated full subset must contain both `nc:Person` and `nc:PersonType`. A software tool that understands how to process NIEM wantlists and schema document subsets (such as the NIEM Schema Subset Generator Tool **[NIEM SSGT]**) can rebuild an accurate schema document subset from a wantlist (and the reverse).

[Definition: NIEM wantlist]

An XML document that represents a complete NIEM schema document subset.

A NIEM wantlist identifies the data component requirements declared by the subset author; it does not identify the data component dependencies required to reconstitute the complete subset. The complete

subset can be computed with the reference schema document set from which the subset was derived.

A wantlist is always associated with a schema document subset. A wantlist may also be associated with a constraint schema document set, because constraint schema documents are often built from a schema document subset. For a simple IEPD, it can sometimes be trivial to identify a single schema document subset. However, this MPD Specification does not prohibit building complex IEPDs that contain schema document sets supported by multiple schema document subsets and associated wantlists. As with other complex cases, the IEPD author is responsible to clearly document the associations between wantlists and schema document sets. In order to maintain a minimal degree of consistency for placement of a wantlist within an IEPD or EIEM:

[Rule 6-1]

A wantlist MUST be a member of the schema document set directory it is associated with. This means it MUST reside with and at the root of the subdirectory that groups and defines its target schema document set.

6.2. Business Rules

For simplicity and consistency, NIEM employs a profile of the XML Schema language **[W3C XML Schema Structures]**, **[W3C XML Schema Datatypes]**. Thus, some constraints on NIEM XML documents cannot be enforced by NIEM. [Constraint schema documents] provide a convenient technique for enforcing some additional constraints. However, even the full XML Schema language cannot validate and enforce all possible constraints that may required on an XML document.

That said, NIEM allows (even encourages) the use of formal or informal *business rules* to supplement MPDs (in particular IEPDs).

[Definition: business rules]

Formal or informal statements that describe business policy or procedure, and in doing so define or constrain some aspect of a process or procedure in order to impose control.

[Business rules] may be represented as informal English statements, or as formally coded machine-readable and processible statements. For example, an IEPD may use a Schematron schema **[ISO Schematron]**, a RelaxNG schema **[ISO RelaxNG]**, or any other formal representation for [business rules].

[Definition: business rule schema]

An artifact that contains [business rules] in a formal representation language with the intent to automatically process them on an XML document to enforce business constraints.

[Definition: Schematron schema]

A [business rule schema] that adheres to **[ISO Schematron]**.

[Definition: RelaxNG schema]

A [business rule schema] that adheres to **[ISO RelaxNG]**.

7. Organization, Packaging, and Other Criteria

An MPD is a logical set of electronic files aggregated and organized to fulfill a specific purpose in NIEM. Directory organization and packaging of an MPD should be designed around major themes in NIEM: reuse, sharing, interoperability, and efficiency.

This rule is also applicable to all MPDs:

[Rule 7-1]

An MPD is packaged as a single compressed archive of files that represents a sub-tree of a file system in standard **[PKZIP]** format. This archive MUST preserve and store the logical directory structure intended by its author.

MPD NIEM schema artifacts must be valid for both XML Schema and NIEM:

[Rule 7-2]

Within an MPD archive, all XSD and XML artifacts MUST be valid against and follow all rules for their respective **[NIEM NDR]** conformance targets; this includes being well-formed and valid XML Schema documents.

NIEM releases, core updates, and domain updates maintain a relatively consistent directory organization **[NIEM Domain Update Specification]**. But there are many ways to organize IEPD and EIEM directories that may depend on a number of factors including (not limited to) business purpose and complexity. For this reason, strict rules for IEPD and EIEM directory structure are difficult to establish. Therefore, IEPD and EIEM authors may create their own logical directory structures subject to the rules of this section.

[Definition: MPD root directory]

The top level file directory relative to all MPD artifacts and subdirectories.

[Rule 7-3]

An MPD archive MUST uncompress (unzip) to a one and only one [MPD root directory].

The foregoing rule ensures that:

- Unpacking an MPD archive will not scatter its contents on a storage device.
- A common starting point always exists to explore or use any MPD.
- mpd-catalog and change log artifacts will always be found in the [MPD root directory] (as a result of **[Rule 4-1]**, above, and **[Rule 4-11]**, above).

7.1. MPD File Name Syntax

As previously stated, the MPD Specification is intended to help facilitate tool support for processing MPDs. Given a tool must process an MPD, providing it basic information about the MPD as early as possible will help to reduce processing time and complexity. So, if the MPD class and version can be easily identified by its file name, then a tool would not have to open the archive and parse the mpd-catalog just to determine this information. Of course, ultimately, to do anything useful, a tool will have to open the MPD archive. However, a standard file name syntax would allow a tool to search through a set of MPDs to find a particular MPD name, version, or class without having to open each. The following rules apply:

[Rule 7-4]

An MPD archive file MUST use file name syntax defined by the regular expression:

```
mpd-filename ::= name '-' version '.' class '.zip'
Where:
        name     ::= alphanum ((alphanum | special)* alphanum)?
        alphanum ::= [a-z0-9]
        special  ::= '.' | '-' | '_'
        version  ::= digit+ ('.' digit+)* (status digit+)?
        digit    ::= [0-9]
        status   ::= 'alpha' | 'beta' | 'rc' | 'rev'
        class    ::= 'rel' | 'cu' | 'du' | 'iepd' | 'eiem'
```

The regular expression notation used in the rule above is from **[W3-XML]** `#sec-notation`.

Alphabetic characters are lower case to reduce the risk of complications across various file systems.

The `status` value options are as defined beneath **[Rule 4-5]**, above.

The `class` value options are defined as follows:

- rel = release
- cu = core update
- du = domain update

- iepd = information exchange package documentation
- eiem = enterprise information exchange model

A valid IEPD file name corresponding to the example in Appendix B, *Example Instance XML Document Catalog*, below, would be: `Planning_Order-1.0.3rev2.iepd.zip`

Checking this Appendix you will find that this example also adheres to the following rule:

[Rule 7-5]

For an MPD, the file name substrings for `name`, `version`, and `class` MUST match exactly the mpd-catalog.xml attribute values for `mpdName`, `mpdVersionID`, and `mpdClassCode` respectively.

In HTTP-based Web Services environments, the MIME type designation of a MPD archive is important to facilitate processing by service consumers.

[Rule 7-6]

When represented on the Internet, an MPD archive SHOULD use the following MIME Type:

```
application/zip+[class]
        where [class] is one member from the value set
        {rel, cu, du, iepd, eiem}.
```

Use of the generic Zip MIME type `application/zip` is allowed, but discouraged. No other MIME types are allowed when representing MPD archives.

7.2. Artifact Links to Other Resources

The **[NIEM NDR]** requires that all namespace references within schema documents can be resolved to the correct local schema document. Recall that there are several approaches to this that may incorporate the mpd-catalog, xml-catalogs, and/or `xs:import schemaLocation` attributes. It is important to understand that the URI scheme defined in Section 4.2.3, *URI Scheme for MPD Artifacts*, above, can only be used only to identify relationships among and provide source links to external schemas being reused. It is not sufficient to allow references or links to such schemas stand in for a physical copy. Thus, all schema artifacts necessary to define, validate, and use an MPD must be physically present within that MPD. In accordance with the **[NIEM NDR]**, if MPD schemas are moved to an operational environment for implementation, validation, or other purposes, then absolute references may replace relative path references when needed. When absolute references to Internet resources are required:

[Rule 7-7]

Absolute references to Internet resources MUST use a well-known transfer protocol (http, https, ftp, ftps) and MUST [resolve] (If applicable, documentation that describes how to [resolve] with

security, account, and/or password issues MUST be included).

Releases, core updates, and domain updates must adhere to packaging rules primarily to enable development tools to process them consistently and efficiently. The NIEM PMO controls the format and documentation for these MPDs and publishes them at `http://release.niem.gov/niem/`. However, many different organizations author IEPDs and EIEMs. As such, they may be distributed, published in repositories (possibly to a limited community), and reused by others. Furthermore, EIEMs are the basis for families of IEPDs. Therefore, it is important that both of these MPD classes are well documented for understanding and use.

[Rule 7-8]

A published IEPD MUST contain all documents necessary to understand it and facilitate its correct implementation.

[Rule 7-9]

A published IEPD MUST link (through its mpd-catalog) to any EIEM it is based on.

Refer to **[Rule 7-6]**, above, for Internet representation of IEPDs or EIEMs.

The **[NIEM NDR]** explains how NIEM employs adapter types to encapsulate and use other standards (e.g., geospatial and emergency management standards) in their native forms that are not NIEM-conformant. Other standards may use `xs:import` without requiring `schemaLocation` attributes (instead, relying only on the namespace value). These standards may also use `xs:include`. This XML Schema construct is disallowed by NIEM. When standards external to NIEM are required within MPDs, the following rule applies:

[Rule 7-10]

Within an MPD, if non-conformant external schema documents are used, then any references from these schema documents to other namespaces MUST [resolve] to local URIs. `schemaLocation` attributes or XML catalogs can be used to ensure resolution.

For the case of non-NIEM-conformant schemas, this rule ensures that all schemas (or corresponding artifacts and namespaces) from external standards required for definition, validation, and use of the MPD are present within the archive.

XML schemas are the heart of MPDs since they formally specify normative structure and semantics for data components. However, in general, an MPD is a closed set of artifacts. This means that all hyperlink references within artifacts should [resolve] to the appropriate artifact.

[Rule 7-11]

Within any artifact of an MPD archive, any direct reference to another resource (i.e., another artifact such as an image, schema, stylesheet, etc.) that is required to process or display an artifact SHOULD exist within the archive at the location specified by that reference.

This means that MPD artifacts, including documentation artifacts, should be complete. For example, if an HTML document contains a hyperlink reference (`href`) to a schema (xsd) or stylesheet (xsl) that is part of the MPD, then the schema file associated with that hyperlink should be present within the MPD; likewise for a sourced (`src`) image. Authors should exercise good judgment with this rule. For example, it does not require an MPD to contain copies of all cited documents from a table of references if it contains hyperlinks to those documents. The key operating words in this rule are: "another resource is required to process or display an artifact SHOULD exist within the archive."

7.3. Duplication of Artifacts

Within an MPD, the replication of files or entire file sets should be avoided. However, replication is allowed if a reasonable rationale exists. In some cases, file replication may make it easier to use, validate, implement, or automatically process an MPD. For example, multiple subsets may overlap with many identical schemas. Yet, it may be easier or even necessary to allow this form of duplication to accommodate a validation tool, rather than removing duplicate schemas, and forcing the tool to search for them. Use xml-catalogs whenever possible.

Appendix A. MPD Catalog XML Schema Document

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema

ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-
and-design-rules/3.0/#ExtensionSchemaDocument"

targetNamespace="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"
    version="3.0"
    xmlns:appinfo="http://release.niem.gov/niem/appinfo/3.0/"
    xmlns:c="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"
    xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"
    xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/"
    xmlns:structures="http://release.niem.gov/niem/structures/3.0/"
    xmlns:term="http://release.niem.gov/niem/localTerminology/3.0/"
    xmlns="http://www.w3.org/2001/XMLSchema">

    <import namespace="http://release.niem.gov/niem/structures/3.0/"/>
    <import namespace="http://release.niem.gov/niem/niem-core/3.0/"/>

<!--
This schema is work-in-progress.  Component names may exist that do not
necessarily conform to the NDR, may yet require review and refinement
to ensure they have all appropriate properties they replaced.
For example, AuthoritativeSource is now type="nc:EntityType".
Nonetheless, we appreciate your feedback. -->

    <annotation>
```

```
      <documentation>
          Model Package Description (MPD) Catalog.
          This schema defines a mpd-catalog.xml artifact
          for NIEM Model Package Descriptions (MPD):
             NIEM releases, core updates, domain updates,
             NIEM Information Exchange Package Documentation (IEPD),
             and NIEM Enterprise Information Exchange Models (EIEM).
          The purpose of this schema is to facilitate consistent declaration
of
          MPD content, metadata, and lineage to process, display, review,
register,
          search, and discover MPDs efficiently. For IEPDs, the mpd-catalog
also
          provides instructions for validating IEPs to schemas.
      </documentation>

      <appinfo>
        <term:LocalTerm term="EIEM"  literal="Enterprise Information Exchange
Model"/>
        <term:LocalTerm term="IANA"  literal="Internet Assigned Numbers
Authority"/>
        <term:LocalTerm term="ID"    literal="Identifier"/>
        <term:LocalTerm term="IEP"   literal="Information Exchange Package"
                                     definition="an instance XML document"/>
        <term:LocalTerm term="IEPD"  literal="Information Exchange Package
                                              Documentation"/>
        <term:LocalTerm term="MIME"  literal="Multipurpose Internet Mail
Extension"/>
        <term:LocalTerm term="MPD"   literal="Model Package Description"/>
        <term:LocalTerm term="OASIS" literal="Organization for the
Advancement
                                              of Structured Information
Standards"/>
        <term:LocalTerm term="POC"   literal="Point of contact"/>
        <term:LocalTerm term="SSGT"  literal="Schema Subset Generation
Tool"/>
        <term:LocalTerm term="URI"   literal="Uniform Resource Identifier"/>
        <term:LocalTerm term="Wantlist" definition="An XML file that
represents a NIEM
                        schema document subset; used by NIEM Schema Subset
Generation
                        Tool to input/output a schema document subset"/>
      </appinfo>

    </annotation>

    <element name="Catalog" type="c:CatalogType">
      <annotation>
        <documentation>An MPD catalog that describes MPD artifacts and
metadata.
        </documentation>
      </annotation>
    </element>

    <complexType name="CatalogType">
      <annotation>
        <documentation>A datatype for an MPD catalog.</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
```

```
            <sequence>
              <element ref="c:MPD"/>
            </sequence>
          </extension>
        </complexContent>
      </complexType>

      <element name="MPD" type="c:MPDType">
        <annotation>
          <documentation>A Model Package Description (MPD).</documentation>
        </annotation>
      </element>

      <complexType name="MPDType">
        <annotation>
          <documentation>A datatype for an MPD.</documentation>
        </annotation>
        <complexContent>
          <extension base="structures:ObjectType">
            <sequence>
             <element ref="nc:DescriptionText"     minOccurs="0"/>
             <element ref="c:MPDInformation"       minOccurs="0"/>
             <element ref="c:IEPConformanceTarget" minOccurs="0"
maxOccurs="unbounded"/>
             <element ref="c:Artifact"             minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
            <attribute ref="c:mpdURI"       use="required"/>
            <attribute ref="c:mpdClassCode" use="required"/>
            <attribute ref="c:mpdName"      use="required"/>
            <attribute ref="c:mpdVersionID" use="required"/>
          </extension>
        </complexContent>
      </complexType>

      <element name="Artifact" abstract="true">
        <annotation>
          <documentation>An file or file set in an MPD.</documentation>
        </annotation>
      </element>

  <!-- File artifact classifiers for a table of contents
  ========================== -->

      <element name="File" type="c:FileType" substitutionGroup="c:Artifact">
        <annotation>
          <documentation>A generic electronic file artifact in an MPD;
          a file stored on a computer system.</documentation>
        </annotation>
        </element>

      <complexType name="FileType">
        <annotation>
          <documentation>A datatype for an MPD file artifact.</documentation>
        </annotation>
        <complexContent>
          <extension base="structures:ObjectType">
            <sequence>
              <element ref="c:RequiredFile"     minOccurs="0"
maxOccurs="unbounded"/>
              <element ref="nc:DescriptionText" minOccurs="0"/>
```

```
          </sequence>
          <attribute ref="c:pathURI"           use="required"/>
          <attribute ref="c:mimeMediaTypeText" use="optional"/>
          <attribute ref="c:externalURI"       use="optional"/>
        </extension>
      </complexContent>
    </complexType>

    <element name="XMLCatalog" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that is an OASIS XML catalog.
</documentation>
      </annotation>
    </element>

    <element name="MPDChangeLog" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that contains a record of the MPD
changes.
        </documentation>
      </annotation>
    </element>

    <element name="MasterDocument" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD master document (readme) artifact.
</documentation>
      </annotation>
    </element>

    <element name="IEPSampleXMLDocument" type="c:FileType"
             substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An example MPD instance XML document or IEP artifact.
        </documentation>
      </annotation>
    </element>

    <element name="BusinessRulesArtifact" type="c:FileType"
             substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that contains business rules
        and constraints on exchange content.</documentation>
      </annotation>
    </element>

    <element name="ExternalSchemaDocument" type="c:FileType"
             substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that is a schema document external to
NIEM.
        </documentation>
      </annotation>
    </element>

    <element name="ReferenceSchemaDocument" type="c:FileType"
             substitutionGroup="c:Artifact">
      <annotation>
```

```
        <documentation>An MPD artifact that is a NIEM reference schema
document.
        </documentation>
      </annotation>
    </element>

    <element name="ExtensionSchemaDocument" type="c:FileType"
             substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that is a NIEM extension schema
document.
        </documentation>
      </annotation>
    </element>

    <element name="SubsetSchemaDocument" type="c:FileType"
              substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that is a subset schema document.
        </documentation>
      </annotation>
    </element>

    <element name="Wantlist" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that represents a NIEM schema subset
        and is used as an import or export for the NIEM SSGT.</documentation>
      </annotation>
    </element>

    <element name="SchematronSchema" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>A Schematron schema document.</documentation>
      </annotation>
    </element>

    <element name="RelaxNGSchema" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>A RelaxNG schema.</documentation>
      </annotation>
    </element>

    <element name="Documentation" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that is a form of explanatory
documentation.
        </documentation>
      </annotation>
    </element>

    <element name="ApplicationInfo" type="c:FileType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD artifact that is used by a software tool
        (e.g., import, export, input, output, etc.).</documentation>
      </annotation>
    </element>
```

```
<!-- For declaring file dependencies
============================================== -->

    <element name="RequiredFile" type="c:FileType">
      <annotation>
        <documentation>An MPD file artifact that another artifact depends on
        and should not be separated from.</documentation>
      </annotation>
    </element>


<!-- Set Artifact Classifiers
================================================== -->

    <complexType name="FileSetType">
      <annotation>
        <documentation>A datatype for an MPD file set artifact.
</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
          <sequence>
            <element ref="nc:DescriptionText" minOccurs="0"/>
            <element ref="c:Artifact"         minOccurs="0"
maxOccurs="unbounded"/>
          </sequence>
          <attribute ref="c:pathURI"     use="required"/>
          <attribute ref="c:externalURI" use="optional"/>
        </extension>
      </complexContent>
    </complexType>

    <element name="SchemaDocumentSet"
             type="c:SchemaDocumentSetType" substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD set artifact that may include schema document
subsets,
        extension and external schema documents, and other supporting
artifacts.
        </documentation>
      </annotation>
    </element>

    <element name="ConstraintSchemaDocumentSet"
             type="c:SchemaDocumentSetType" substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD set artifact that may include constraint schema
        document subsets, and associated extension and external schema
documents,
        and other supporting artifacts.</documentation>
      </annotation>
    </element>

   <complexType name="SchemaDocumentSetType">
      <annotation>
        <documentation>A datatype for an MPD set artifact that may include
schema
        document subsets, extension schema documents, and external schema
documents.
```

```
        </documentation>
      </annotation>
      <complexContent>
        <restriction base="c:FileSetType">
          <sequence>
            <element ref="nc:DescriptionText"  minOccurs="0"/>
            <element ref="c:XMLCatalog"/>
            <element ref="c:Wantlist"          minOccurs="0"/>
            <element ref="c:XMLSchemaDocument" minOccurs="0"
                                               maxOccurs="unbounded"/>
          </sequence>
        </restriction>
      </complexContent>
    </complexType>

    <element name="ReferenceSchemaDocumentSet"
             type="c:ReferenceSchemaDocumentSetType"
substitutionGroup="c:Artifact">
      <annotation>
        <documentation>An MPD set artifact that may include constraint schema
        document subsets, and associated extension and external schema
documents,
        and other supporting artifacts.</documentation>
      </annotation>
    </element>

    <complexType name="ReferenceSchemaDocumentSetType">
      <annotation>
        <documentation>A datatype for an MPD set artifact
        that is a reference schema document set.</documentation>
      </annotation>
      <complexContent>
        <restriction base="c:FileSetType">
          <sequence>
            <element ref="nc:DescriptionText" minOccurs="0"/>
            <element ref="c:ReferenceSchemaDocument" maxOccurs="unbounded"/>
          </sequence>
        </restriction>
      </complexContent>
    </complexType>


<!-- Primitives
================================================================== -->

    <attribute name="mpdURI" type="anyURI">
      <annotation>
        <documentation>A globally unique identifier (URI) for an MPD.
</documentation>
      </annotation>
    </attribute>

    <attribute name="mpdName" type="c:MPDNameSimpleType">
      <annotation>
        <documentation>A descriptive label or title for an MPD.
</documentation>
      </annotation>
    </attribute>

    <simpleType name="MPDNameSimpleType">
      <annotation>
```

```
        <documentation>A datatype for an MPD name, label, or title.
</documentation>
      </annotation>
      <restriction base="token">
        <pattern value="[a-z]([-_]?[a-z0-9]+)*"/>
      </restriction>
    </simpleType>

    <attribute name="mpdVersionID" type="c:MPDVersionIDSimpleType">
      <annotation>
        <documentation>An identifier that distinguishes releases of a given
MPD.
        </documentation>
      </annotation>
    </attribute>

    <simpleType name="MPDVersionIDSimpleType">
      <annotation>
        <documentation>A datatype for an identifier that distinguishes
releases
        of a given MPD.</documentation>
      </annotation>
      <restriction base="token">
        <pattern value="[0-9]+(\.[0-9]+)*((alpha|beta|rc|rev)[0-9]+)?"/>
      </restriction>
    </simpleType>

    <attribute name="mpdClassCode" type="c:MPDClassCodeSimpleType">
      <annotation>
        <documentation>A classification for an MPD; values are drawn
        from the set {rel, cu, du, iepd, eiem}.</documentation>
      </annotation>
    </attribute>

    <simpleType name="MPDClassCodeSimpleType">
      <annotation>
        <documentation>A datatype for the classification of an MPD.
</documentation>
      </annotation>
        <restriction base="token">
            <enumeration value="rel">
                <annotation>
                  <documentation>release</documentation>
                </annotation>
            </enumeration>
            <enumeration value="cu">
                <annotation>
                  <documentation>core update</documentation>
                </annotation>
            </enumeration>
            <enumeration value="du">
                <annotation>
                  <documentation>domain update</documentation>
                </annotation>
            </enumeration>
            <enumeration value="iepd">
                <annotation>
                  <documentation>information exchange package documentation
                  </documentation>
                </annotation>
            </enumeration>
```

```
            <enumeration value="eiem">
                <annotation>
                  <documentation>enterprise information exchange
model</documentation>
                </annotation>
            </enumeration>
        </restriction>
</simpleType>

    <attribute name="pathURI" type="anyURI">
      <annotation>
      <documentation>A URI for the pathname of a local artifact relative to
      the MPD root directory.</documentation>
      </annotation>
    </attribute>

    <attribute name="externalURI" type="anyURI">
      <annotation>
        <documentation>A globally unique identifier (URI) for an artifact
        in another MPD that is reused by this MPD.</documentation>
      </annotation>
    </attribute>

    <attribute name="mimeMediaTypeText" type="string">
      <annotation>
        <documentation>A classification for an MPD file
        artifact from the IANA MIME media classes:
        http://www.iana.org/assignments/media-types.</documentation>
      </annotation>
    </attribute>

    <complexType name="RelationshipType">
      <annotation>
        <documentation>A datatype for a reference to another MPD related to
this MPD.
        </documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
          <sequence>
            <element ref="nc:DescriptionText" minOccurs="0"/>
          </sequence>
          <attribute ref="c:relationshipCode" use="required"/>
          <attribute ref="c:resourceURI"      use="required"/>
        </extension>
      </complexContent>
    </complexType>

    <attribute name="resourceURI"   type="anyURI">
      <annotation>
        <documentation>A globally unique identifier (URI) for another
        MPD or document to which this MPD relates.</documentation>
      </annotation>
    </attribute>

    <attribute name="relationshipCode" type="c:RelationshipCodeSimpleType">
      <annotation>
        <documentation>A classification or reason for the connectedness
between
        this MPD and the resource referenced in resourceURI.</documentation>
      </annotation>
```

```
    </attribute>

    <simpleType name="RelationshipCodeSimpleType">
      <annotation>
        <documentation>A datatype for a classification
        of the relationship between MPDs.</documentation>
      </annotation>
        <restriction base="token">
            <enumeration value="version_of">
              <annotation>
                <documentation>A relationshipCode value for indicating that
this MPD
                is a different version of the MPD referenced in resourceURI.
This
                code value is only needed in cases where significant name
changes
                might obscure the relationship to the previous version.  For
example,
                NIEM Justice 4.1 is a version of GJXDM 3.0.3.</documentation>
                </annotation>
            </enumeration>
            <enumeration value="specializes">
              <annotation>
                <documentation>A relationshipCode value for indicating that
this
                MPD is a generalization of the MPD referenced in resourceURI.
                This value is the inverse of specializes.</documentation>
              </annotation>
            </enumeration>
            <enumeration value="generalizes">
              <annotation>
                <documentation>A relationshipCode value for indicating that
this MPD
                is a generalization of the MPD referenced in resourceURI.
This
                value is the inverse of specializes.</documentation>
                </annotation>
            </enumeration>
            <enumeration value="supersedes">
                <annotation>
                  <documentation>A relationshipCode value for indicating that
this
                  MPD replaces the MPD referenced in resourceURI.
</documentation>
                </annotation>
            </enumeration>
            <enumeration value="deprecates">
                <annotation>
                  <documentation>A relationshipCode value for indicating that
content
                  in this MPD is preferred over content in the MPD referenced
in
                  resourceURI; and at some time in the future will supersede
the
                  MPD referenced in resourceURI.</documentation>
                </annotation>
            </enumeration>
            <enumeration value="adapts">
                <annotation>
                  <documentation>A relationshipCode value for indicating that
this
```

```
                    MPD is an adaptation of the MPD referenced in resourceURI.
                    </documentation>
                </annotation>
            </enumeration>
            <enumeration value="updates">
                <annotation>
                  <documentation>A relationshipCode value for indicating that
                  this MPD is an incremental update to the resource
referenced in
                  resourceURI.  Used by a core or domain update to identify
the
                  domain schema in a NIEM release being incrementally updated
                  (not replaced).</documentation>
                </annotation>
            </enumeration>
            <enumeration value="conforms_to">
                <annotation>
                  <documentation>A relationshipCode value for indicating that
this
                  MPD conforms to the specification or standard referenced in
                  resourceURI.</documentation>
                </annotation>
            </enumeration>
        </restriction>
    </simpleType>


<!-- IEP Conformance Targets
================================================== -->

    <element name="IEPConformanceTarget" type="c:IEPConformanceTargetType">
      <annotation>
        <documentation>A class or category of IEPs which has a set of
validity
        constraints and a unique identifier. Every IEP is an instance of one
or
        more IEP Conformance Targets.</documentation>
      </annotation>
    </element>

    <complexType name="IEPConformanceTargetType">
      <annotation>
        <documentation>A datatype for a class or category of IEP, which
        has a set of validity constraints and a unique identifier.
</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
          <sequence>
            <element ref="nc:DescriptionText"     minOccurs="0"/>
            <element ref="c:ValidityConstraint"   minOccurs="0"
                                                  maxOccurs="unbounded"/>
            <element ref="c:IEPSampleXMLDocument" minOccurs="0"
                                                  maxOccurs="unbounded"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <element name="ValidityConstraint" abstract="true">
      <annotation>
```

```
        <documentation>A rule or instructions for validating an IEP
candidate.
        </documentation>
      </annotation>
    </element>


<!-- Validity Constraints
========================================================= -->

    <element name="ValidToXPath" type="c:XPathType"
            substitutionGroup="c:ValidityConstraint">
      <annotation>
        <documentation>A validity constraint that indicates that the given
XPath
        expression, evaluated against an artifact, has an effective boolean
value
        of TRUE.</documentation>
      </annotation>
    </element>

    <complexType name="XPathType">
      <annotation>
        <documentation>A datatype for an XPath expression.</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
          <sequence>
            <element ref="nc:DescriptionText" minOccurs="0"/>
          </sequence>
          <attribute ref="c:xPathText" use="required"/>
        </extension>
      </complexContent>
    </complexType>

    <attribute name="xPathText" type="string">
      <annotation>
        <documentation>An XPath expression.</documentation>
      </annotation>
    </attribute>

    <element name="XMLSchemaValid"
             type="c:XMLSchemaValidationType"
             substitutionGroup="c:ValidityConstraint">
      <annotation>
        <documentation>A validity constraint that indicates that an artifact
must
        be locally XML Schema valid against an XML schema described by an XML
Catalog
        file, starting with a given validation root.</documentation>
      </annotation>
    </element>

    <complexType name="ValidationWithValidationRootType">
      <annotation>
        <documentation>A datatype for a base for a validation that may start
with
        a validation root.</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
```

```
          <sequence>
            <element ref="nc:DescriptionText"    minOccurs="0"/>
            <element ref="c:ValidationRootXPath" minOccurs="0"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <complexType name="XMLSchemaValidationType">
      <annotation>
        <documentation>A datatype for an XML Schema validation constraint,
indicating
        an XML Schema against which an artifact may be validated, as well as
a
        description of validation roots for assessment of validity.
</documentation>
      </annotation>
      <complexContent>
        <extension base="c:ValidationWithValidationRootType">
          <sequence>
            <element ref="c:XMLCatalog"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <element name="ValidationRootXPath" type="c:XPathType">
      <annotation>
        <documentation>An XPath expression that identifies a seqence of zero
or more
        validation root elements for a validity constraint on an XML
document.
        </documentation>
      </annotation>
    </element>

    <element name="SchematronValid"
             type="c:SchematronValidationType"
             substitutionGroup="c:ValidityConstraint">
      <annotation>
        <documentation>A validity constraint that indicates that an artifact
must
        be valid against the rules carried by a Schematron file, starting
with
        the identified validation roots.</documentation>
      </annotation>
    </element>

    <complexType name="SchematronValidationType">
      <annotation>
        <documentation>A datatype for a Schematron validation constraint,
indicating
        a Schematron schema document against which an artifact may be
validated
        as well as a description of the validation roots for assessment of
validity.
        </documentation>
      </annotation>
      <complexContent>
        <extension base="c:ValidationWithValidationRootType">
          <sequence>
```

```
            <element ref="c:SchematronSchema"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <element name="RelaxNGValid" type="c:RelaxNGValidationType"
        substitutionGroup="c:ValidityConstraint">
      <annotation>
        <documentation>A validity constraint that indicates that an artifact
must
        be valid against the rules carried by a RelaxNG schema.
</documentation>
      </annotation>
    </element>

    <complexType name="RelaxNGValidationType">
      <annotation>
        <documentation>A datatype for a RelaxNG validation constraint,
indicating a
        RelaxNG schema document against which an artifact may be validated,
as well
        as a description of the validation roots for assessment of validity.
        </documentation>
      </annotation>
      <complexContent>
        <extension base="c:ValidationWithValidationRootType">
          <sequence>
            <element ref="c:RelaxNGSchema"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <element name="HasDocumentElement" type="c:QualifiedNamesType"
        substitutionGroup="c:ValidityConstraint">
      <annotation>
        <documentation>A validity constraint that indicates that an artifact
has
        a document element with a name that is one of the given qualified
names.
        </documentation>
      </annotation>
    </element>

    <element name="HasElement" type="c:QualifiedNamesType"
        substitutionGroup="c:ValidityConstraint">
      <annotation>
        <documentation>A validity constraint that indicates that an artifact
        has an element with a name that is one of the given qualified names.
        </documentation>
      </annotation>
    </element>

    <complexType name="QualifiedNamesType">
      <annotation>
        <documentation>A datatype for a set of qualified names.
</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
```

```
            <sequence>
              <element ref="nc:DescriptionText" minOccurs="0"/>
            </sequence>
            <attribute ref="c:qualifiedNameList" use="required"/>
          </extension>
        </complexContent>
      </complexType>

      <attribute name="qualifiedNameList" type="c:QualifiedNameListSimpleType">
        <annotation>
          <documentation>A list of qualified names.</documentation>
        </annotation>
      </attribute>

      <simpleType name="QualifiedNameListSimpleType">
        <annotation>
          <documentation>A simple datatype denoting a list of qualified names.
          </documentation>
        </annotation>
        <list itemType="QName"/>
      </simpleType>

      <element name="ConformsToConformanceTarget"
type="c:ConformanceTargetType"
             substitutionGroup="c:ValidityConstraint">
        <annotation>
          <documentation>A validity constraint that indicates that an artifact
          must conform to the given conformance target.</documentation>
        </annotation>
      </element>

      <complexType name="ConformanceTargetType">
        <annotation>
          <documentation>A datatype for identifying and describing a
conformance target.
          </documentation>
        </annotation>
        <complexContent>
          <extension base="structures:ObjectType">
            <sequence>
              <element ref="nc:DescriptionText" minOccurs="0"/>
            </sequence>
            <attribute ref="c:conformanceTargetURI" use="required"/>
          </extension>
        </complexContent>
      </complexType>

      <attribute name="conformanceTargetURI" type="anyURI">
        <annotation>
          <documentation>A URI for a conformance target.</documentation>
        </annotation>
      </attribute>

      <element name="ConformsToRule" type="c:TextRuleType"
          substitutionGroup="c:ValidityConstraint">
        <annotation>
          <documentation>A validity constraint that indicates that an artifact
          must conform to the given text rule, drafted in a human language.
          </documentation>
        </annotation>
      </element>
```

```
    <complexType name="TextRuleType">
      <annotation>
        <documentation>A datatype for a rule drafted
        in a human language.</documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
          <sequence>
            <element ref="nc:DescriptionText" minOccurs="0"/>
            <element ref="c:RuleText"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <element name="RuleText" type="nc:TextType">
      <annotation>
        <documentation>A rule written in a human language.</documentation>
      </annotation>
    </element>


<!-- Metadata
================================================================== -->

    <element name="MPDInformation" type="c:MPDInformationType">
      <annotation>
        <documentation>A set of descriptive data about an MPD.
</documentation>
      </annotation>
    </element>

    <complexType name="MPDInformationType">
      <annotation>
        <documentation>A datatype for a set of descriptive data about an MPD.
        </documentation>
      </annotation>
      <complexContent>
        <extension base="structures:ObjectType">
          <sequence>
            <element ref="c:AuthoritativeSource" minOccurs="0"/>
            <element ref="c:CreationDate"        minOccurs="0"/>
            <element ref="c:LastRevisionDate"    minOccurs="0"/>
            <element ref="c:StatusText"          minOccurs="0"/>
            <element ref="c:Relationship"        minOccurs="0"
maxOccurs="unbounded"/>
            <element ref="c:KeywordText"         minOccurs="0"
maxOccurs="unbounded"/>
            <element ref="c:DomainText"          minOccurs="0"
maxOccurs="unbounded"/>
            <element ref="c:PurposeText"         minOccurs="0"
maxOccurs="unbounded"/>
            <element ref="c:ExchangePatternText" minOccurs="0"
maxOccurs="unbounded"/>
            <element ref="c:ExchangePartnerName" minOccurs="0"
maxOccurs="unbounded"/>
            <element ref="c:ExtendedInformation" minOccurs="0"
maxOccurs="unbounded"/>
          </sequence>
        </extension>
```

```
      </complexContent>
    </complexType>

    <element name="ExtendedInformation" abstract="true">
      <annotation>
        <documentation>User-defined descriptive data about an MPD.
</documentation>
      </annotation>
    </element>

    <element name="AuthoritativeSource" type="nc:EntityType">
      <annotation>
              <documentation>An official sponsoring or authoring
              organization responsible for an MPD.</documentation>
      </annotation>
    </element>

<!-- c:AuthoritativeSource should use NIEM 3.0 schema components as follows
...

<c:AuthoritativeSource>
(type="nc:EntityType")
        <nc:EntityOrganization>                         (or
<nc:EntityPerson>)
                <nc:OrganizationName>                   (type="nc:TextType")
                <nc:OrganizationLocation>
                        <nc:Address>
                                <nc:AddressFullText>    (type="nc:TextType")
                <nc:OrganizationPrimaryContactInformation>
                        <nc:ContactResponder>
                        <nc:PersonName>
                                <nc:PersonFullName>     (type="nc:TextType")
                        <nc:ContactEmailID>             (type="xs:string")
                        <nc:ContactTelephoneNumber>
                                <nc:TelephoneNumberRepresentation>
(substitutionGroup)
                        <nc:ContactWebsiteURI>          (type="xs:anyURI")
-->

    <element name="CreationDate" type="date">
      <annotation>
        <documentation>A date this MPD was published.</documentation>
      </annotation>
    </element>

    <element name="LastRevisionDate" type="date">
      <annotation>
        <documentation>A date the latest changes to an MPD were published
        (i.e., CreationDate of previous version).</documentation>
      </annotation>
    </element>

    <element name="StatusText"   type="string">
      <annotation>
         <documentation>A description of the current state of this MPD
         in development; may also project future plans for the MPD.
         </documentation>
      </annotation>
    </element>

    <element name="Relationship" type="c:RelationshipType">
```

```
      <annotation>
        <documentation>A reference to another MPD related to this MPD.
        </documentation>
      </annotation>
    </element>

    <element name="KeywordText"  type="string">
      <annotation>
        <documentation>A common alias, term, or phrase that would help
        to facilitate search and discovery of this MPD.</documentation>
      </annotation>
    </element>

    <element name="DomainText"   type="string">
      <annotation>
        <documentation>A description of the environment or NIEM Domain
        in which this MPD is applicable or used.</documentation>
      </annotation>
    </element>

    <element name="PurposeText"  type="string">
      <annotation>
        <documentation>A description of the intended usage and reason
        for which an MPD exists.</documentation>
      </annotation>
    </element>

    <element name="ExchangePatternText" type="string">
      <annotation>
        <documentation>A description of a transactional or design pattern
used
        for this IEPD (generally, only applicable to IEPDs).</documentation>
      </annotation>
    </element>

    <element name="ExchangePartnerName" type="string">
      <annotation>
        <documentation>A name of an entity or organization that uses this MPD
        (generally, only applicable to IEPDsP).
        </documentation>
      </annotation>
    </element>

</schema>
```

Appendix B. Example Instance XML Document Catalog

[TBD]

```
<?xml version="1.0" encoding="US-ASCII"?>
<c:Catalog
   xmlns:c="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"
   xmlns:structures="http://release.niem.gov/niem/structures/3.0/"
   xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/"
   xmlns:ns="http://example.org/namespace">
  <c:MPD
     c:mpdURI="http://example.org/myIEPD/"
```

```
      c:mpdClassCode="iepd"
      c:mpdName="Speeding Ticket"
      c:mpdVersionID="1.0"
      >
    <c:Metadata>
      <c:CreationDate>2013-09-18</c:CreationDate>
      <c:AuthoritativeSource>
      </c:AuthoritativeSource>
    </c:Metadata>
    <c:IEPConformanceTarget structures:id="message">
      <c:XMLSchemaValid>
        <nc:DescriptionText>MUST be valid to this XML Schema.
</nc:DescriptionText>
        <c:ValidationRootXPath c:xPathText="//ns:ParkingTicket"/>
        <c:XMLCatalog c:pathURI="xml-catalog.xml"/>
      </c:XMLSchemaValid>
      <c:SchematronValid>
        <c:SchematronSchema c:pathURI="rules.sch"/>
      </c:SchematronValid>
      <c:ValidToXPath c:xPathText="//xml-catalog.xml"/>
    </c:IEPConformanceTarget>
    <c:MasterDocument
       c:pathURI="master-document.html">
    </c:MasterDocument>
  </c:MPD>
</c:Catalog>
```

Appendix C. MPD Artifacts

[TBD]

Appendix D. Guidance for IEPD Directories (non-normative)

NIEM releases, core updates, and domain updates generally follow a consistent directory organization. When employing release and updates within IEPDs and EIEMs whether as-is or as subsets, users are encouraged to maintain their original directory structures. However, aside from applicable rules previously stated in the preceding sections, there are no normative rules for organizing directories within IEPDs or EIEMs.

Guidance for directory structuring may be useful to authors, especially in the case of a relatively simple IEPD or EIEM with a single schema document subset, and a few extension, exchange, and external schema documents. The following has been common non-normative practice for IEPD directories:

- Create a root directory for the IEPD from the name and version identifier of the IEPD. For example `my_iepd-3.2rev4`.
- Per **[Rule 4-1]**, above, and **[Rule 4-11]**, above, the `mpd-catalog.xml` and the change log must reside in [MPD root directory].
- If extending the [MPD catalog document], then per **[Rule 4-2]**, above `mpd-catalog-extension-xml-catalog.xml` MUST reside in the same directory as the `mpd-catalog.xml` it supports. `mpd-catalog-extension.xsd` can be anywhere in the MPD

because `mpd-catalog-extension-xml-catalog.xml` will locate it. However, recommend all these files be co-located in the [MPD root directory] for visibility.

- Maintain a NIEM schema subset organized as generated by the Schema Subset Generation Tool (SSGT). The reason is that SSGT generates an [XML catalog document] that correctly correlates each schema document namespace to its corresponding relative pathname (local URI). SSGT also provides the correct `xs:schemaLocation` attribute values for each `xs:import` element.
- As generated by SSGT, a NIEM subset will be contained in a `/niem` subdirectory. Put this subdirectory into the [MPD root directory].
- If a [NIEM wantlist] exists for a subset, put it into the `/niem` subdirectory to maintain its association with its subset.
- If using a [reference schema document] set (such as a release), maintain as originally downloaded. This will be the same as SSGT organizes a subset.
- If derived from a schema subset, maintain a constraint schema set organized like the subset from which it was derived (for the same reason as above).
- Maintain extension schema documents in a subdirectory of the [MPD root directory] with a name that begins with the substring `extension`.
- Maintain external standard schema documents that are not built into NIEM (and therefore are not in the subset under `/niem`) in a subdirectory with a name that begins with the substring `external`.
- Maintain each constraint schema document set (or all constraint schema documents if appropriate) in a subdirectory with a name that begins with the substring `niem-constraint`. If the constraint schema is derived from a schema subset, maintain the same subdirectory organization as the subset.
- Maintain Schematron schema documents in a subdirectory with a name that begins with the substring `schematron`.
- Maintain sample IEPs in a subdirectory with a name that begins with the substring `iep-sample`. This subdirectory should also contain any XML stylesheets (XSL) used with the sample instances.
- Maintain documentation in a subdirectory with a name that begins with the substring `documentation`. Create additional documentation subdirectories inside this one as needed.
- Maintain tool-specific artifacts (inputs, outputs, imports, exports, etc.) in a subdirectory with a name that begins with the substring `application-info`.

Obviously, there are many other ways to organize for more complex business requirements in which multiple releases, subsets, constraint sets, core updates, and domain updates are employed in a single IEPD or EIEM. Regardless of directory organization and file naming, an IEPD or EIEM author should always configure all IEP conformance targets using the MPD catalog `IEPConformanceTarget` element and the appropriate validation artifacts (such as XML catalogs, Schematron schemas, RelaxNG schemas, etc.).

The guidance above results in an IEPD directory structure that appears below. Filenames within the `extension`, `external`, `exchange`, `schematron`, and `iep-sample` subdirectories are non-normative examples. Authors are free to assign names for such files according to their own requirements (if they do not violate the rules in this specification for files such as `mpd-catalog.xml` or `changelog.*`).

/my_iepd-3.2rev4 (root directory of IEPD archive)

```
mpd-catalog.xml
mpd-catalog-extension-xml-catalog.xml
mpd-catalog-extension.xsd
changelog.*
master-document.*

/niem
        /adaptors
        /appinfo
        /codes
        /conformanceTargets
        /domains
        /external
        /localTerminology
        /niem-core
        /proxy
        /structures
        wantlist.xml
        xml-catalog.xml

/extension
        query.xsd
        response.xsd
        extension1.xsd
        extension2.xsd
        ...
        xml-catalog.xml

/external
        /stix
        /icism
        ...
        xml-catalog.xml

/niem-constraint
        /adaptors
        /appinfo
        /codes
        /conformanceTargets
        ...
        wantlist.xml
        xml-catalog.xml

/schematron
        business-rules1.sch
        business-rules2.sch
        ...

/iep-sample
        query.xml
        request.xml
        ...

/application-info
        ... (tool inputs, outputs, etc.)

/documentation
        ... (human readable documentation)
```

Appendix E. Acronyms and Abbreviations

Acronym / Abbreviation	Literal or Definition
ASCII	American Standard Code for Information Interchange
BIEC	Business Information Exchange Component
CSV	Comma Separated Value (file format)
CU	Core update
DU	Domain updater
EBV	Effective Boolean Value
EIEM	Enterprise Information Exchange Model
GIF	Graphic Interchange Format
GML	Geospatial Markup Language
HTML	Hyper Text Markup Language
IEP	Information Exchange Package
IEPD	Information Exchange Package Documentation
JPG	Joint Photographic (Experts) Group
LEXS	Logical Entity Exchage Specifications
MPD	Model Package Description
NDR	Naming and Design Rules
NIEM	National Information Exchange Model
NTAC	NIEM Technical Architecture Committee
PDF	Portable Document Format
PMO	Program Management Office
PNG	Portable Network Graphic
QName	Qualified Name (XML Schema: qualified by a namespace prefix)
RAR	Roshal Archive; a compressed archive file format named for its developer, Eugene Roshal
SSGT	Schema Subset Generation Tool
UML	Unified Modeling Language
UPA	Unique Particle Attribution
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformation

Appendix F. References

[FEA Data Reference Model]: *The Federal Enterprise Architecture Data Reference Model*, Version 1.0, September 2004. Available from `http://xml.gov/documents/completed/DRMv1.pdf`. A more recent DRM Version 2.0, 17 November 2005 is available from `http://www.whitehouse.gov/omb/assets/egov_docs/DRM_2_0_Final.pdf`

[GJXDM IEPD Guidelines]: *GJXDM Information Exchange Package Documentation Guidelines*, Version 1.1, Global XML Structure Task Force (GXSTF), 2 March 2005. Available from `http://it.ojp.gov/documents/global_jxdm_IEPD_guidelines_v1_1.pdf`

[ISO 11179-4]: *ISO/IEC 11179-4 Information Technology — Metadata Registries (MDR) — Part 4: Formulation of Data Definitions Second Edition,* 15 July 2004. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c035346_ISO_IEC_11179-4_2004(E).zip`.

[ISO 11179-5]: *ISO/IEC 11179-5:2005, Information technology — Metadata registries (MDR) — Part 5: Naming and identification principles*. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c035347_ISO_IEC_11179-5_2005(E).zip`.

[ISO RelaxNG]: *Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*, ISO/IEC 19757-2:2008, Second Edition, 15 December 2008. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c052348_ISO_IEC_19757-2_2008(E).zip`. See also `http://relaxng.org`.

[ISO Schematron]: *Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron*, ISO/IEC 19757-3:2006(E), First Edition, 1 June 2006. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip`.

[Logical Entity Exchange Specifications]: *Logical Entity Exchange Specifications*, Version 4.0, 27 July 2011. Available from `http://130.207.211.107/content/downloads`.

[NIEM BIEC]: *Business Information Exchange Components (BIEC)*, Version 1.0, NIEM Technical Architecture Committee (NTAC), March 2011. Available from `http://reference.niem.gov/niem/guidance/business-information-exchange-components/1.0/`.

[NIEM Conformance]: *NIEM Conformance*, Version 1.0, NIEM Technical Architecture Committee (NTAC), 15 September 2008. Available from `http://reference.niem.gov/niem/specification/conformance/1.0/`.

[NIEM Conformance Target Specification]: *NIEM Conformance Target Specification*, Version 1.0, NIEM Technical Architecture Committee (NTAC), 15 January 2013. Available from `http://reference.niem.gov/niem/specification/conformance-target/1.0/`.

[NIEM Concept of Operations]: *NIEM Concept of Operations*, Version 0.5, NIEM Program Management Office, 9 January 2007. Available from `http://reference.niem.gov/niem/guidance/concept-of-operations/`.

[NIEM Domain Update Specification]: *NIEM Domain Update Specification*, Version 1.0, NIEM Technical Architecture Committee (NTAC), 5 November 2010. Available from http://reference.niem.gov/niem/specification/domain-update/1.0/.

[NIEM High-Level Tool Architecture]: *NIEM High-Level Tool Architecture*, Version 1.1, NIEM Technical Architecture Committee, 1 December 2008. Available from http://reference.niem.gov/niem/specification/high-level-tool-architecture/1.1/.

[NIEM High-Level Version Architecture]: *NIEM High Level Version Architecture (HLVA)*, Version 1.0, NIEM Technical Architecture Committee, 2008. Available from http://reference.niem.gov/niem/specification/high-level-version-architecture/1.0/.

[NIEM IEPD Requirements]: *Requirements for a National Information Exchange Model (NIEM) Information Exchange Package Documentation (IEPD) Specification*, Version 2.1, June 2006. Available from http://reference.niem.gov/niem/guidance/iepd-requirements/2.1/.

[NIEM Implementation Guidance]: "NIEM Implementation Guide", NIEM Program Management Office. Available from https://www.niem.gov/program-managers/Pages/implementation-guide.aspx.

[NIEM Introduction]: *Introduction to the National Information Exchange Model (NIEM)*, Version 0.3, NIEM Program Management Office, 12 February 2007. Available from http://reference.niem.gov/niem/guidance/introduction/.

[NIEM MPD Specification]: *NIEM Model Package Description (MPD) Specification*, Version 3.0, NIEM Technical Architecture Committee (NTAC), day month [TBD] 2014. Available from http://reference.niem.gov/niem/specification/model-package-description/3.0/.

[NIEM NDR]: *NIEM Naming and Design Rules (NDR)*, Version 3.0, NIEM Technical Architecture Committee (NTAC), 31 October 2008. Available from http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/.

[NIEM SSGT]: NIEM Schema Subset Generation Tool (SSGT). Available from http://tools.niem.gov/niemtools/ssgt/index.iepd.

[NIEM User Guide]: *NIEM User Guide*, Volume 1, U.S. Department of Justice, Office of Justice Programs, (date unknown). Available from http://reference.niem.gov/niem/guidance/user-guide/vol1/.

[XML Catalogs]: *XML Catalogs*, Organization for the Advancement of Structured Information Standards (OASIS) Standard v1.1, 7 October 2005. Available from https://www.oasis-open.org/committees/download.php/14809/std-entity-xml-catalogs-1.1.html.

[RAR]: http://win-rar.com

[RFC2119 Key Words]: Bradner, S., *Key words for use in RFCs to Indicate Requirement*

Levels, IETF RFC 2119, March 1997. Available from http://www.ietf.org/rfc/rfc2119.txt.

[RFC3986 URI]: Berners-Lee, T., et al., *Uniform Resource Identifier (URI): Generic Syntax*, Request for Comments 3986, Network Working Group, January 2005. Available from http://tools.ietf.org/html/rfc3986.

[W3-EXI]: *Efficient XML Interchange (EXI) Format*, Version 1.0, W3C Recommendation, 10 March 2011. Available from http://www.w3.org/TR/2011/REC-exi-20110310/.

[W3-XML]: *Extensible Markup Language (XML)*, Version 1.0, Fifth Edition, W3C Recommendation 26 November 2008. Available from http://www.w3.org/TR/2008/REC-xml-20081126/.

[W3-XML-InfoSet]: *XML Information Set*, Second Edition, W3C Recommendation 4 February 2004. Available from http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[W3-XML-Namespaces]: *Namespaces in XML*, Second Edition, World Wide Web Consortium 16 August 2006. Available from http://www.w3.org/TR/2006/REC-xml-names-20060816/.

[W3C XML Schema Datatypes]: *XML Schema Part 2: Datatypes*, Second Edition, W3C Recommendation 28 October 2004. Available from http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[W3C XML Schema Structures]: *XML Schema Part 1: Structures*, Second Edition, W3C Recommendation 28 October 2004. Available from http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[W3C XPath 2.0]: *XML Path Language (XPath) 2.0*, Second Edition, W3C Recommendation 14 December 2010. Available from http://www.w3.org/TR/2010/REC-xpath20-20101214/.

[XSLT v1.0]: *XSL Transformations (XSLT)*, Version 1.0, W3C Recommendation 16 November 1999. Available from http://www.w3.org/TR/1999/REC-xslt-19991116.

[XSLT v2.0]: *XSL Transformations (XSLT)*, Version 2.0, W3C Recommendation 23 January 2007. Available from http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[PKZIP]: *APPNOTE.TXT - .ZIP File Format Specification*, Version: 6.3.2, Revised: 28 September 2007, Copyright (c) 1989 - 2007 PKWare Inc. Available from http://www.pkware.com/documents/casestudies/APPNOTE.TXT.